СОДЕРЖАНИЕ

Номер 1, 2020

CONTENTS

No. 1, 2020

ПРОГРАММИРОВАНИЕ, 2020, № 1, с. 3–14

КОМПЬЮТЕРНАЯ ГРАФИКА И ВИЗУАЛИЗАЦИЯ

УДК 004.42

ИДЕНТИФИКАЦИЯ И РАСЧЕТ ТРАЕКТОРИЙ ДИНАМИЧЕСКИХ ОБЪЕКТОВ ПО СТЕРЕОИЗОБРАЖЕНИЯМ

© 2020 г. В. А. Бобков[a],*, А. П. Кудряшов[a],**

[a] *Институт автоматики и процессов управления Дальневосточного отделения РАН*
690041 Владивосток, ул. Радио, 5, Россия

**E-mail: bobkov@dvo.ru*

***E-mail: kudryashovA@dvo.ru*

Поступила в редакцию 03.06.2019 г.
После доработки 17.06.2019 г.
Принята к публикации 23.06.2019 г.

Предложен модифицированный подход к решению задачи восстановления траекторий динамических объектов в сцене по стереоизображениям, основанный на применении расширенного точечного представления объектов, метода визуальной одометрии и оригинальной алгоритмической базы. Описаны алгоритмы идентификации объектов и вычислительная схема поэтапной обработки данных. Приведены результаты вычислительных экспериментов на модельных сценах.

DOI: 10.31857/S0132347420010021

1. ВВЕДЕНИЕ

В настоящей статье представлены результаты дальнейшего развития подхода к решению проблемы 3D SLAM в динамической сцене, опубликованного авторами в [1]. Некоторое представление о состоянии дел в этом направлении можно получить из работ [2–12], краткий обзор которых сделан в [1]. Была отмечена недостаточная степень универсальности существующих решений и не всегда приемлемая эффективность программно-алгоритмических средств для практических приложений.

В [1] был изложен подход, основанный на применении визуальной одометрии, использовании точечной модели объектов и реализации оригинальной алгоритмической базы для идентификации и обработки статических и динамических объектов (ДО) сцены без априорного знания о “статике”. Под ДО подразумеваются движущиеся объекты с непредсказуемыми моментами времени появления и исчезновения в сцене. Термин “статика” объединяет все неподвижные объекты в сцене. Однако полученные оценки эффективности первой реализации подхода указали на необходимость совершенствования разработанной алгоритмической базы. Основной недостаток заключался в генерации недостаточного числа точек в формируемых 3D моделях динамических объектов, что сказывалось на точности локализации объектов и качестве реконструируемых поверхностей объектов.

Поэтому в настоящей статье предлагается усовершенствованная вычислительная схема и алгоритмическая база с реализацией новых решений, направленных на повышение эффективности данного подхода к решению задачи SLAM для динамических сцен. К основным элементам новизны следует отнести использование плотного множества 3D точек, расширенного за счет измерений 3D сканера (или его виртуального аналога на базе стереоизображений) и возможность различать динамические объекты с неизменяемой и изменяемой формой.

2. ОБЩИЙ ПОДХОД

Решается задача идентификации и восстановления движения динамических объектов в априори неизвестной обстановке по изображениям, фиксируемым стереокамерой, установленной на автономном роботе. Предполагается, что в дополнение к оптической информации поступают и данные от 3D сканера (или от его виртуального аналога – программной реализации на базе стереокамеры [13]).

Предлагаемый подход к решению указанной задачи основывается на:

– применении точечного представления объектов сцены;

– применении визуального метода навигации для вычисления относительных перемещений камеры и объектов, и расчета их траекторий движения;

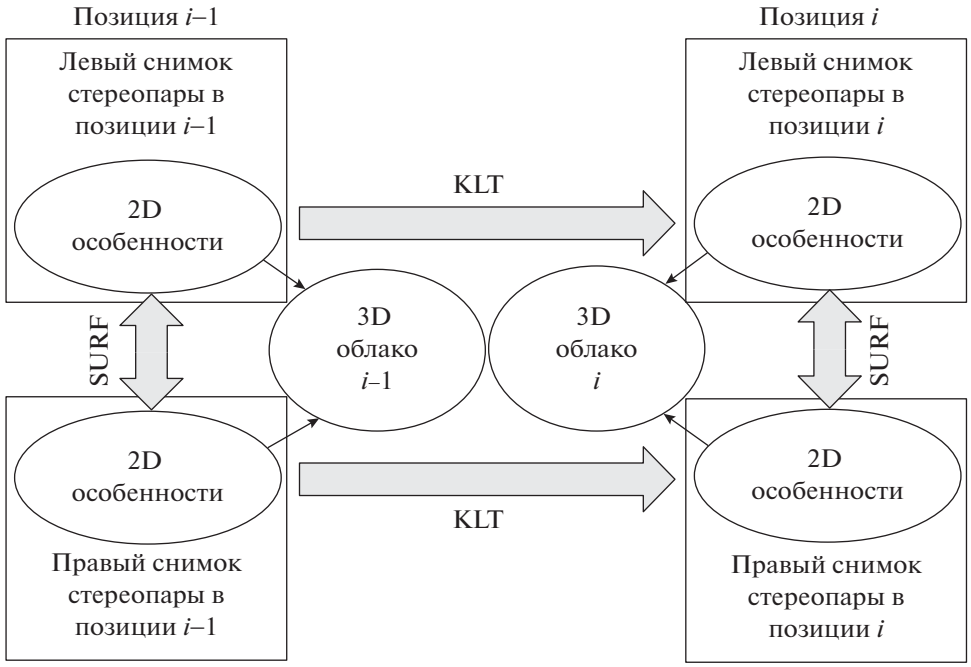


Рис. 1. Выделение единого сопоставленного множества особенностей на изображениях двух стереопар. Формирование 2-х 3D облаков.

– разработке алгоритмической базы, направленной на решение ключевых задач рассматриваемой проблемы, а именно:

– выделение ДО и генерация их 3D точечных моделей,

– выделение статической части сцены и вычисление траектории движения камеры,

– идентификация ДО, сформированных в разные моменты времени (в разных позициях траектории движения камеры),

– вычисление траекторий собственного движения ДО на основе декомпозиции комплексного движения объектов, фиксируемого камерой (движение камеры плюс собственное движение объекта).

Предлагаемое в настоящей работе развитие ранее описанного в [1] подхода состоит в реализации следующих решений:

– обрабатывается плотное множество 3D точек, получаемых, как за счет данных от 3D сканера (или его виртуального аналога), так и за счет сопоставления особенностей на последовательных во времени снимках стереопар (в начальной версии использовалось только множество, построенное на сопоставленных особенностях);

– в дополнение к ранее разработанному алгоритму формирования 3D точечных моделей ДО на основе критерия “жесткости” предлагается алгоритм “затравочного” типа с использованием критерия “связности точек”;

– задача идентификации и вычисления траектории движения решается также и применительно к ДО, изменяющим свою форму при движении.

Для восстановления движения камеры и объектов сцены применяется метод визуальной навигации (авторские реализации его модификаций описаны в [14–16]). Метод следует классической схеме визуальной одометрии:

– установление соответствий 2D точечных особенностей на двух парах снимков, соответствующим двум соседним расчетным позициям на траектории. Для поиска соответствий точечных особенностей на левом и правом снимке стереопары используется широко применяемый детектор SURF (Speeded Up Robust Features) [17]. А для отслеживания точечных особенностей на двух снимках, соответствующих двум соседним расчетным позициям на траектории, применяется известный трекер KLT (Kanade–Lucas–Tomasi feature tracker) [18];

– построение методом триангуляции лучей по сопоставленному на стереопаре текущей позиции множеству точечных особенностей соответствующего множества 3D точек (3D облако). Аналогичным образом строится 3D облако для стереопары предшествующей позиции. Каждое 3D облако описывается в локальной системе координат (СК) позиции (рис. 1);

– вычисление по сопоставленным 3D облакам матрицы геометрического преобразования, опи-

сывающего относительное перемещение камеры/робота (6 DOF (degrees of freedom)) в локальной СК.

Плотное 3D точечное представление видимой части сцены в каждый момент времени формируется, как было отмечено выше, за счет объединения двух множеств 3D точек, принадлежащих объектам, – точек, получаемых обработкой изображений стереопар (3D облако), и 3D точек, непосредственно измеряемых 3D сканером, или его виртуальным аналогом. Заметим, что первому множеству 3D точек соответствует сопоставленное множество в предшествующей позиции (построенное по сопоставленным особенностям). Это соответствие используется в алгоритме установления связи/соответствия между ДО соседних позиций. Реализуемое плотное точечное представление дает возможность наряду с другими методами выделения объектов в видеопотоке применять и метод, основанный на критерии "связности", учитывающим непрерывность поверхности объекта.

3. ПОСТРОЕНИЕ ТОЧЕЧНОЙ МОДЕЛИ ДИНАМИЧЕСКОГО ОБЪЕКТА

Задача построения 3D точечных моделей статических и динамических объектов сцены решается на исходном неструктурированном множестве 3D точек (видимых в данной позиции камеры). Все множество 3D точек в текущей позиции i, в которой должны быть сформированы точечные модели статических и динамических объектов, состоит из трех независимо формируемых 3D множеств (см. рис. 2). Здесь:

G_{i-1} – множество 2D особенностей, сопоставленных на изображениях стереопары позиции $(i-1)$ (на предыдущем шаге). $G_{i-1,i}$ множество 2D особенностей в позиции i, прослеженных от G_{i-1}. G_i-множество 2D особенностей, сопоставленных на изображениях стереопары позиции i (на текущем шаге). $G_{i,i+1}$ – множество 2D особенностей в позиции $i+1$, прослеженных от G_i. $M^{i-1}(G_{i-1})$ – множество 3D точек, построенное в СК позиции $(i-1)$ по особенностям G_{i-1} (сформировано на предыдущем шаге). $M^i_{i-1,i}(G_{i-1,i})$ – множество 3D точек, построенное в СК позиции i по особенностям $G_{i-1,i}$. 3D точкам этого множества сопоставлены точки множества $M^{i-1}(G_{i-1})$ в силу выполняемого на текущем шаге прослеживания/сопоставления соответствующих 2D особенностей. $M^i(G_i)$ – множество 3D точек, построенное на текущем шаге в СК позиции i по 2D особенностям G_i. $M^{i+1}_{i,i+1}(G_{i,i+1})$ – множество 3D точек, которое должно быть построено на следующем шаге в СК позиции $(i+1)$ по особенностям $G_{i,i+1}$. 3D точкам этого множества будут сопоставлены точки множества $M^i(G_i)$ в силу прослеживания/сопоставления соответствующих 2D особенностей на шаге $(i+1)$. R_i – плотное множество 3D точек, полученное "дальномером" на текущем шаге в позиции i.

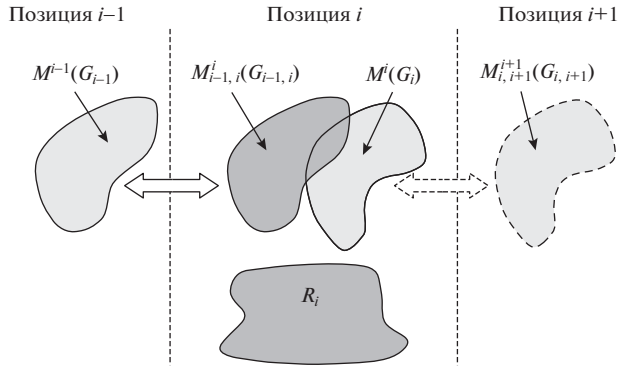


Рис. 2. Объединенное множество исходных 3D точек на текущем шаге: $M^i_{i-1,i} + R_i + M^i$.

Под СК позиции понимается система координат камеры съемки, или, что эквивалентно, СК, связанная с роботом, на котором камера жестко закреплена.

Заметим, что сопоставленные 3D множества точек в соседних позициях являются разными координатными представлениями одного и того же множества точек реальных объектов, видимых камерой в разные моменты времени (из двух соседних позиций).

Следует также отметить, что согласно предлагаемой схеме обработки, формируемое на текущем шаге множество 2D особенностей G_i сохраняется, и используется двояким образом. Во-первых, на текущем шаге по сопоставленным на стереопаре особенностям строится множество 3D точек $M^i(G_i)$. А, во-вторых, на следующем шаге (в позиции $(i+1)$) эти особенности прослеживаются с помощью KLT и используются для построения множества 3D точек $M^{i+1}_{i,i+1}$, которое будет сопоставлено с M^i. Таким образом, устанавливается связь позиций i и $(i+1)$, что необходимо для идентификации ДО на протяжении всего времени их движения в поле зрения камеры.

Построения 3D точечных моделей статических и динамических объектов сцены выполняется в несколько этапов. Вначале генерируются т.н. "затравочные" точки, каждая из которых принадлежит какому-то объекту. При этом, возможно, какие-то "затравочные" точки могут принадлежать одному и тому же объекту. Затем выполняется сортировка точек исходного множества по

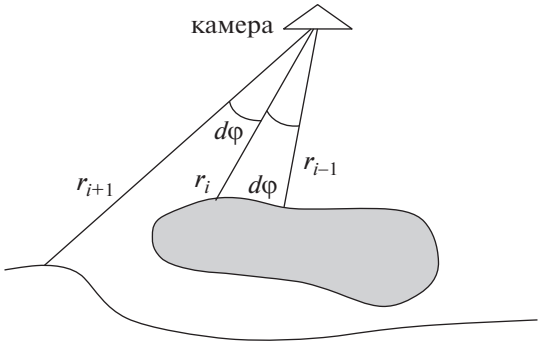


Рис. 3. Проверка принадлежности 3D точки объекту по критерию связности.

группам (отдельным подмножествам точек), порожденным "затравочными" точками, т.е. генерация 3D точечных моделей объектов. Отбор точек осуществляется применением двух критериев, – критерия "связности" и критерия "жесткости", о которых говорится ниже. Тот факт, что одному объекту могут принадлежать несколько "затравочных" точек, и, соответственно, несколько отдельных групп точек, не противоречит этой методике, поскольку такие группы объединяются в единую 3D модель объекта на основе критерия "сходства движения". Поскольку сформированные группы точек могут относиться, как к статическим, так и к динамическим объектам, в последующем изложении будем использовать понятие динамической группы (ДГ), являющейся точечным представлением ДО или его части.

3.1. Алгоритм генерации затравочных точек

В простом для реализации варианте затравочные точки могут выбираться на регулярной пространственной решетке, ограничивающей сцену. Этого достаточно с учетом вышесказанного о возможности объединения групп точек, принадлежащих одному объекту.

3.2. Алгоритм формирования множеств точек по критерию жесткости

Критерий "жесткости" строим исходя из двух положений: а) расстояние между двумя точками недеформируемого/жесткого объекта сохраняется при его движении; б) в качестве точки заведомо принадлежащей конкретному объекту будем рассматривать его "затравочную" точку. Тогда отбор тех точек из исходного множества, которые принадлежат данному объекту, можно осуществить с помощью следующего критерия:

пусть $P_{\text{затр}}$ – затравочная точка, а P тестируемая точка в первом 3D облаке,

$P'_{\text{затр}}$ и P' их сопоставленные образы во втором 3D облаке, а

$$r1 = P_{\text{затр}} - P, \quad r2 = P'_{\text{затр}} - P'.$$

Если $r1 = r2$ (с определенной пороговой погрешностью), то точка P принадлежит объекту, представленному затравочной точкой $P_{\text{затр}}$. Чтобы исключить использование порога, реализуется другая форма критерия: выбирается $\min|r1 - r2|$ после вычисления этой разности для всех "затравочных" точек.

3.3. Алгоритм формирования групп точек по критерию связности

Согласно алгоритму "затравочного" типа обработка точек исходного множества 3D точек начинается с затравочной точки (заведомо принадлежащей объекту). Для каждой пары соседних точек (очередная анализируемая точка и предшествующая точка, получившая статус принадлежности к группе) проверяется выполнение критерия "связности". Согласно критерию, сравниваются расстояния от камеры до текущей и до предшествующей соседней точки (рис. 3):

Пусть $\varepsilon(r_k, r_{k-1}) = |dr|/|d\varphi|$, где $d\varphi$ – смещение по углу между лучами для двух соседних точек, а $|dr| = |r_k - r_{k-1}|$. Тогда величина coherence $= \varepsilon2\,(r_{i+1}, r_i)/\varepsilon1\,(r_{i-1}, r_i)$ будет характеризовать связность/непрерывность поверхности. Если coherence > пороговой величины, то имеет место разрыв поверхности, т.е. выход на границу объекта. В противном случае текущая точка принадлежит группе (объекту).

3.4. Алгоритм объединения групп точек со сходным движением

Сформированные алгоритмом связности группы точек в текущей позиции i могут относиться не только к разным объектам, но и к одному и тому же объекту. Поэтому для такого рода групп необходимо решать задачу их объединения в одну группу. Это важно для решения задачи обнаружения точек всех статических объектов в сцене и для восстановления движения каждого из ДО.

Исходим из предположения, что в каждой группе присутствуют точки множества R_i и множества $M^i_{i-1,i}$. Алгоритм объединения основывается на применении критерия "жесткости", который позволяет оценить сходство движения точек. Заметим, что алгоритм применим только к сопоставленным в двух позициях точкам (точки множества R_i не сопоставлены). Обозначим множество сформированных на текущем шаге групп через G. Для каждой группы $g_k \in G$ ищем на множестве $\{G - g_k\}$ группы со схожим движением, применяя указанный критерий. Пусть группа $g_l \in G$ проверяется на предмет объединения с группой g_k. Тогда

согласно критерию “жесткости” рассматриваются две пары точек (рис. 4). В качестве первой точки в первой паре берется точка P^{g_k} в группе g_k, принадлежащая множеству $M^i_{i-1,i}$. В качестве второй точки берется точка P^{g_l} в группе g_l, также принадлежащая множеству $M^i_{i-1,i}$. Во второй паре в качестве первой точки берется точка $P^{g_k'}$, являющаяся прообразом точки P^{g_k}. А в качестве второй точки берется соответственно точка $P^{g_l'}$, являющаяся прообразом точки P^{g_l}. Обе точки принадлежат множеству $M^{i-1}_{i-1,i}$, заданному в СК позиции $(i-1)$. Построим критерий “жесткости”:

$$r1 = |P^{g_k} - P^{g_l}|, \quad r2 = |P^{g_k'} - P^{g_l'}|.$$

Если $r1 = r2$ (с определенной пороговой погрешностью) для нескольких точек P^{g_l}, то считаем, что обе группы принадлежат “статике” или одному ДО, и, соответственно, могут быть объединены.

Если в группе g_k отсутствуют точки множества $M^i_{i-1,i}$, то считаем, что эта группа точек принадлежит новому ДО. Таким образом, используя сопоставленные пары 3D облаков предыдущей и текущей позиции, определяем одинаковые группы со схожими движениями. После работы “алгоритма объединения” должны остаться только группы, каждая из которых полностью представляет ДО.

4. СОПОСТАВЛЕНИЕ ДИНАМИЧЕСКИХ ГРУПП ТОЧЕК СОСЕДНИХ ПОЗИЦИЙ

Поскольку формирование динамических групп (ДГ) выполняется независимо при обработке каждой позиции траектории камеры, необходимо решать задачу идентификации каждой группы по отношению к группам предыдущей позиции. Т.е. необходимо устанавливать соответствие двух ДГ, фиксируемых/наблюдаемых в разные моменты времени, но представляющих один и тот же объект. Это требуется для расчета траектории движения каждого объекта на протяжении всего времени и для построения его анимационной 3D модели. Трудность заключается в том, что в каждой позиции формирование ДГ точек, принадлежащих ДО, выполняется безотносительно к предшествующей позиции. Дополнительная трудность – непредсказуемое появление новых ДО и исчезновение из сцены прежних ДО, а также необходимость различать недеформируемые и деформируемые ДО. В рассматриваемых ниже алгоритмах установления соответствия между ДГ соседних позиций используется локальная матрица геометрического преобразования, описывающая относительное совместное движение объекта и

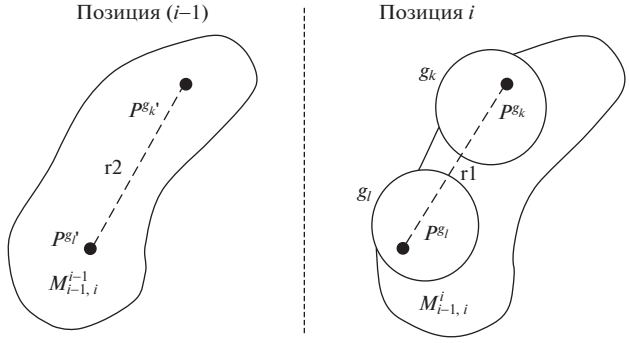


Рис. 4. Объединение группы g_l с группой g_k по критерию “жесткости”.

камеры. Здесь следует заметить, что вычисление этой матрицы делается по-разному для недеформируемых и деформируемых ДО.

4.1. Вычисление локальной матрицы преобразования (относительное движение) для групп точек, представляющих недеформируемые динамические объекты

Для каждой ДГ, сформированной в позиции i, вычисляем методом визуальной одометрии матрицу преобразования координат $H^{joint,id_{gr}}_{i-1,i}$ точек этой ДГ из СК позиции $(i-1)$ в СК позиции i. Здесь id_{gr} – идентификатор (номер) ДГ. Заметим, что $H^{joint,id_{gr}}_{i-1,i}$ – матрица комплексного преобразования, поскольку она отражает результат одновременного движения камеры и ДО (представленного данной ДГ). Движению объекта относительно камеры можно сопоставить симметричное движение камеры относительно объекта. Поэтому матрицу локального комплексного преобразования (вычисленного по сопоставленным 3D облакам этого объекта в соседних позициях) можно представить как:

$$H^{joint,id_{gr}}_{i-1,i} = H^{camera}_{i-1,i} \cdot H^{id_{gr}}_{i-1,i}, \qquad (1)$$

здесь $H^{camera}_{i-1,i}$ – матрица, описывающая движение камеры (предварительно вычислена по множеству точек статических объектов), а $H^{id_{gr}}_{i-1,i}$ – матрица, описывающая собственное относительное перемещение данного ДО.

Следует отметить, что первый способ вычисления матрицы $H^{joint,id_{gr}}_{i-1,i}$ основывается на использовании в качестве исходных данных точек ДГ, которые принадлежат сопоставленным множествам $M^{i-1}_{i-1,i}$ и $M^i_{i-1,i}$ (как это делалось в первой версии системы [1]). Однако при наличии в ДГ точек плотного множества R (полученных дальномером) возможно использование расширенного ис-

ходного множества точек для более точного вычисления матрицы. В этом случае для вычисления матрицы применяется алгоритм ICP, работающий на не сопоставленных явным образом исходных множествах точек, поскольку, как это было выше указано, множества R формируются в каждой позиции независимо друг от друга.

4.2. Вычисление локальной матрицы преобразования для групп точек, представляющих деформируемые динамические объекты

В случае ДО с изменяемой во времени формой (деформируемых), его точки могут осуществлять разнонаправленные собственные движения (в отличие от недеформируемых). Соответственно, мы не можем вычислить локальную матрицу (которая объединяет движение камеры и собственное движение объекта) $H_{i-1,i}^{joint,id_{gr}}$ методом визуальной одометрии, как это делается для недеформируемых объектов. Поэтому будем рассматривать/моделировать движение деформируемого ДО, как движение его центра масс (далее будем называть центральной точкой), определяемого по множеству принадлежащих данной ДГ точек, сопоставленных на шаге $[i-1, i]$. Тогда относительное перемещение центральной точки ДГ из позиции $(i-1)$ в позицию i можно описывать матрицей, объединяющей матрицу движения камеры и матрицу переноса (описывающую собственное перемещение центральной точки). Но, в данном случае (в отличие от вышеописанного случая недеформируемых ДО) матрица, описывающая собственное движение ДО, содержит только вектор перемещения центра масс (a, b, c):

$$H_{i-1,i}^{id_{gr}} = \begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ a & b & c & 1 \end{matrix}$$ – матрица, описывающая собственное перемещение центральной точки на неизвестный вектор (a, b, c) за время перемещения камеры из позиции $(i-1)$ в позицию i.

Пусть:

$P_{center}^{CS^{i-1},id_{gr}}$ – центральная точка ДГ с именем id_{gr}, измеренная в СК$^{i-1}$ (по множеству сопоставленных точек);

$P_{center}^{CS^{i-1},id_{gr}matched} = P_{center}^{CS^{i-1},id_{gr}} \cdot H_{i-1,i}^{camera}$ – образ центральной точки $P_{center}^{CS^{i-1},id_{gr}}$ в СК^i, полученный с учетом перемещения камеры (без учета собственного движения);

$P_{center}^{CS^{i},id_{gr}}$ – центральная точка, измеренная в СК^i (по множеству сопоставленных точек).

Тогда элементы a, b, c в матрице $H_{i-1,i}^{id_{gr}}$ определяем как компоненты вектора смещения ΔP (a, b, c) $= P_{center}^{CS^{i},id_{gr}} - P_{center}^{CS^{i-1},id_{gr}matched}$

Таким образом, мы сформировали комплексную матрицу

$$H_{i-1,i}^{joint,id_{gr}} = H_{i-1,i}^{camera} \cdot H_{i-1,i}^{id_{gr}},$$

связывающую координаты центральной точки ДГ (деформируемого ДО) в СК$^{i-1}$ и в СК^i.

4.3. Алгоритм идентификации динамических групп точек в текущей позиции

Как было отмечено выше, для прослеживания всех ДГ на протяжении всего времени наблюдения/съемки необходимо идентифицировать каждую из ДГ на каждом очередном шаге. Это означает, что для сформированной в новой позиции ДГ требуется установить ее соответствие с одной из ДГ, сформированной в предшествующей позиции. Поскольку сформированные в каждой позиции ДГ уже разделены на два типа – недеформируемые и деформируемые группы, то поиск соответствия для сформированной в новой позиции недеформируемой ДГ осуществляется на множестве ДГ этого же типа предшествующей позиции. То же самое справедливо и для деформируемых ДГ. Такое соответствие для недеформируемой ДГ, сформированной в позиции (i), может быть установлено двумя способами. Первый способ – соответствие устанавливается алгоритмом, основанным на сравнении вычисляемых интегральных оценок геометрической близости точек этой ДГ к пространственным оболочкам ДГ в позиции $(i-1)$. Второй способ – соответствие устанавливается проверкой наличия сопоставленных 3D точек в двух анализируемых ДГ. Недостаток алгоритма оценки геометрической близости – вычислительная трудоемкость. Поэтому был выбран способ решения, основанный на выявлении в сравниваемых ДГ сопоставленных 3D точек, которые служат “маркерами”. Наличие таких точек в обеих группах, принадлежащих, соответственно, сопоставленным множествам $M_{i-1,i}^{i-1}$ и $M_{i-1,i}^{i}$, подтверждает, что обе точечные модели представляют один и тот же ДО. Поэтому алгоритм идентификации некоторой ДГ в текущей позиции i по отношению к предыдущей позиции состоит в последовательной проверке всех ДГ в предыдущей позиции на наличие в обеих группах общих сопоставленных точек, принадлежащих указанным множествам. Иллюстрация к работе алгоритма показана на рис. 5. В случае успешного сопоставления тестируемой ДГ в позиции i ей присваивается идентификатор ее прообраза в позиции $(i-1)$. Одновременно по принципу унаследования свойств определяется и тип ДО – “недеформиру-

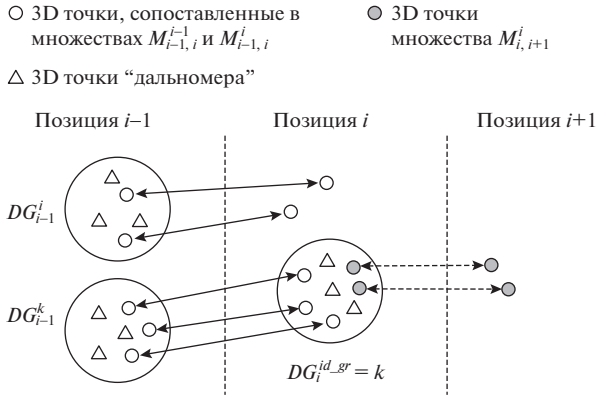


Рис. 5. Идентификация ДГ. Группа точек ДО $DG^{id_{gr}}_{i}$, сформированная в позиции i, сопоставлена с DG^{k}_{i-1} в позиции $(i-1)$ и не сопоставлена с DG^{l}_{i-1}.

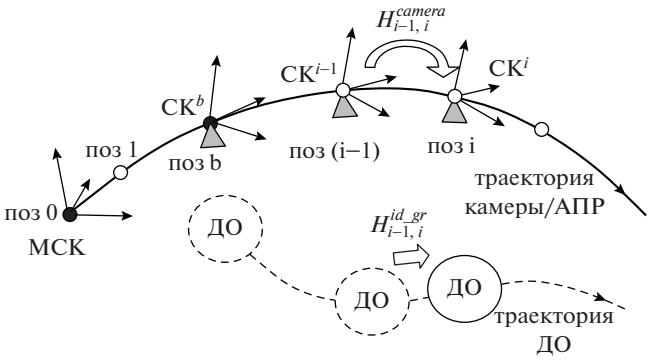


Рис. 6. Вычисление матриц геометрического преобразования, определяющих собственное движение динамического объекта. Позиция b – начало движения ДО в сцене.

емый" или "деформируемый". Если такого сопоставления не обнаружено, то это значит, что ДГ относится к новому ДО. Заметим, что возможно применение обоих рассмотренных критериев (алгоритмов) для повышения достоверности идентификации ДГ.

Для нового ДО необходимо определить его тип. Определение типа делается алгоритмом, основанным на применении к точкам сформированной ДГ критерия "жесткости". Точки ДГ проверяются критерием на сходство движения. Для проверки используются точки, принадлежащие сопоставленным множествам $M^{i-1}_{i-1,i}$ и $M^{i}_{i-1,i}$.

При сходстве движения точек квалифицируем ДО как "недеформируемый", в противном случае – как "деформируемый".

5. О ВЫЧИСЛЕНИИ ТРАЕКТОРИЙ И РЕКОНСТРУКЦИИ ДИНАМИЧЕСКИХ ОБЪЕКТОВ

Каждый ДО характеризуется своим временем жизни в сцене от момента времени его появления в сцене t_b до момента времени выхода его из сцены t_e. С моментом времени t_b связана начальная система координат данного объекта. В этой СК будет описываться собственное движение объекта, безотносительно к движению камеры. Также в этой СК будет выполняться генерация полной 3D модели этого объекта – т.е. в нее будут помещены все полученные виды объекта с последующим объединением. В качестве такой начальной СК объекта выберем (без ущерба общности) локальную СК камеры/робота в позиции b. Координаты точек ДО из его начальной позиции b (а также из других позиций) могут быть преобразованы согласно методу визуальной одометрии в СК начальной позиции траектории робота (t_0). В свою очередь, при наличии преобразования, связывающего СК начальной позиции траектории робота с некоторой внешней системой координат, можно получить координаты объектов сцены во внешней системе координат. Назовем эту внешнюю СК мировой системой координат (МСК). В качестве МСК часто используют (без ущерба общности) СК робота в начальной позиции, предполагая, что упомянутое выше преобразование во внешнюю СК известно. Собственное относительное перемещение ДО за время $[t_{i-1}, t_i]$ будем оценивать вычислением локальной матрицы геометрического преобразования на шаге $[(i-1), i]$ (см. (1)).

Вычисление траектории ДО. В качестве траектории ДО можно рассматривать траекторию движения любой его точки из начальной позиции b (момент времени t_b) в конечную позицию e (момент времени t_e). Построить такую траекторию можно вычисляя последовательно локальные матрицы геометрического преобразования $H^{id_{gr}}_{i-1,i}$ (рис. 6), которые определяют собственное движение ДО в интервалах времени $t_b - t_1$, $t_1 - t_2$, ... $t_{i-1} - t_i$, ... $t_{e-1} - t_e$.

Из (1) следует:

$$H^{id_{gr}}_{i-1,i} = H^{joint,id_{gr}}_{i-1,i} \cdot (H^{camera}_{i-1,i})^{-1} \qquad (2)$$

То есть, по вычисленному движению камеры ($H^{camera}_{i-1,i}$) и совместному движению объекта u камеры ($H^{joint,id_{gr}}_{i-1,i}$) мы можем вычислить матрицу геометрического преобразования координат $H^{id_{gr}}_{i-1,i}$, определяющую собственное локальное перемещение ДО. Тогда преобразование 3D координат точки $P^{id_{gr}}_{l}$ объекта id_{gr} на шаге $[(i-1), i]$ определяется соотношением:

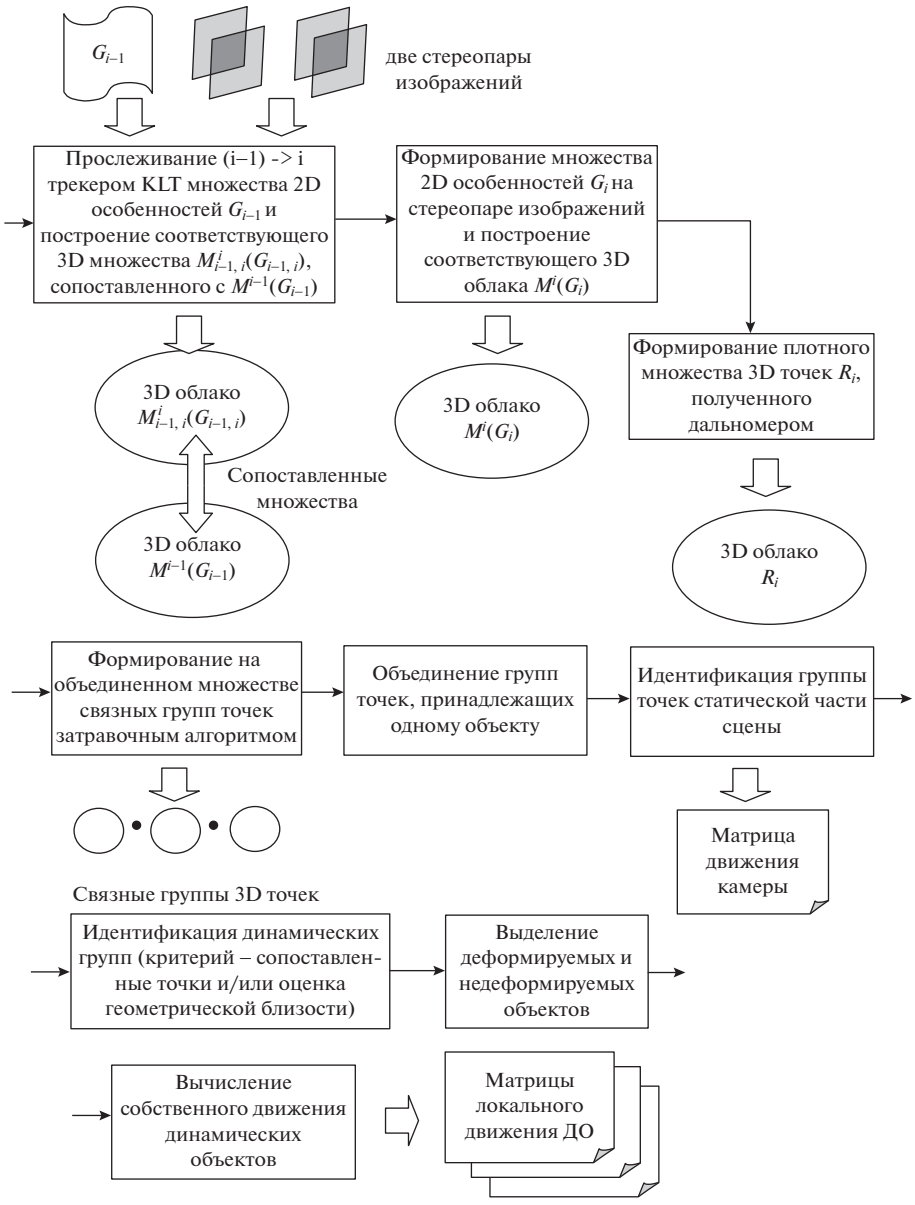


Рис. 7. Вычислительная схема обработки данных на текущем шаге траектории.

$$P_l^{t_i,id_{gr}} = P_l^{t_{i-1},id_{gr}} \cdot H_{i-1,i}^{id_{gr}}, \quad (3)$$

здесь $P_l^{t_i,id_{gr}}$ – координаты точки l объекта id_{gr} в момент времени t_i.

Матрица результирующего преобразования ДО с именем id_{gr} за время $[t_b, t_i]$ (собственное относительное движение) получается перемножением соответствующих локальных матриц:

$$H_{b,i}^{id_{gr}} = \prod_{k=b}^{k=i-1} H_{k,k+1}^{id_{gr}}, \quad (4)$$

где b – позиция камеры в момент появления ДО с именем id_{gr}.

С учетом (4) координаты точек объекта id_{gr} в момент времени t_i связаны с координатами точек в начальной СК этого объекта матрицей $H_{b,i}^{id_{gr}}$:

$$P_l^{t_i,id_{gr}} = P_l^{t_b,id_{gr}} \cdot H_{b,i}^{id_{gr}} \quad (5)$$

Для помещения всех ДО и траекторий их движения в единую СК, которой является МСК, необходимо иметь преобразование из начальной СК ДО в МСК. Поскольку в качестве МСК мы рассматриваем локальную систему координат камеры в начальный момент времени, то $H_{WCS,b}^{camera} = \prod_{k=0}^{k=b-1} H_{k,k+1}^{camera}$. Со-

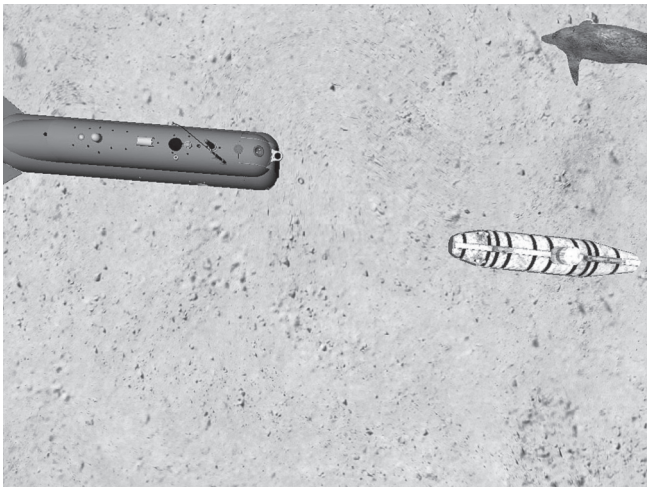


Рис. 8. Снимок подводной сцены в одной из позиций траектории АПР: в поле зрения камеры находятся 3 динамических объекта.

ответственно, координаты рассматриваемой точки $P_l^{t_i,id_{gr}}$ в МСК могут быть получены как:

$$P_l^{WCS,id_{gr}} = P_l^{t_i,id_{gr}} \cdot (H_{WCS,b}^{camera})^{-1}, \quad (6)$$

3D реконструкция ДО. Чтобы построить 3D модель объекта по видам объекта, полученным из разных позиций камеры (в разные моменты времени), необходимо совместить эти виды в СК начальной позиции этого ДО. Это реализуется с помощью следующих операций:

1) Из каждой позиции i траектории камеры видимые точки $P_n^{CK^i,id_{gr}}$ объекта id_{gr}, построенные по стереопаре в CK^i, помещаются в СК начальной позиции этого ДО, т.е. в СК позиции b камеры (момент появления в сцене этого ДО). Это делается с помощью матрицы $H_{b,i}^{joint,id_{gr}}$, которая учитывает и движение камеры, и преобразование координат точек ДО, связанное с его перемещением:

$$P_n^{CK^{i,b},id_{gr}} = P_n^{CK^i,id_{gr}} \cdot (H_{b,i}^{joint,id_{gr}})^{-1}$$ здесь

$P_n^{CK^{i,b},id_{gr}}$ – 3D точка с номером n объекта id_{gr}, преобразованная из CK^i в CK^b.

Матрица $H_{b,i}^{joint,id_{gr}}$ согласно методу визуальной одометрии получается перемножением соответствующих локальных матриц:

$H_{b,i}^{joint,id_{gr}} = \prod_{k=b}^{k=i-1} H_{k,k+1}^{joint,id_{gr}}$, где локальная матрица $H_{k,k+1}^{joint,id_{gr}}$ определяет совместное движение камеры и ДО из позиции k в позицию $(k+1)$.

2) Чтобы получить координаты точек объектов всех видов (ракурсов) в МСК необходимо применить преобразование $(H_{WCS,b}^{camera})^{-1}$, где $H_{WCS,b}^{camera} = \prod_{k=0}^{k=b-1} H_{k,k+1}^{camera}$.

Таким образом, возможность пересчета координат точек каждого ДО в МСК позволяет реконструировать их траектории и 3D модели в едином координатном пространстве мировой системы координат.

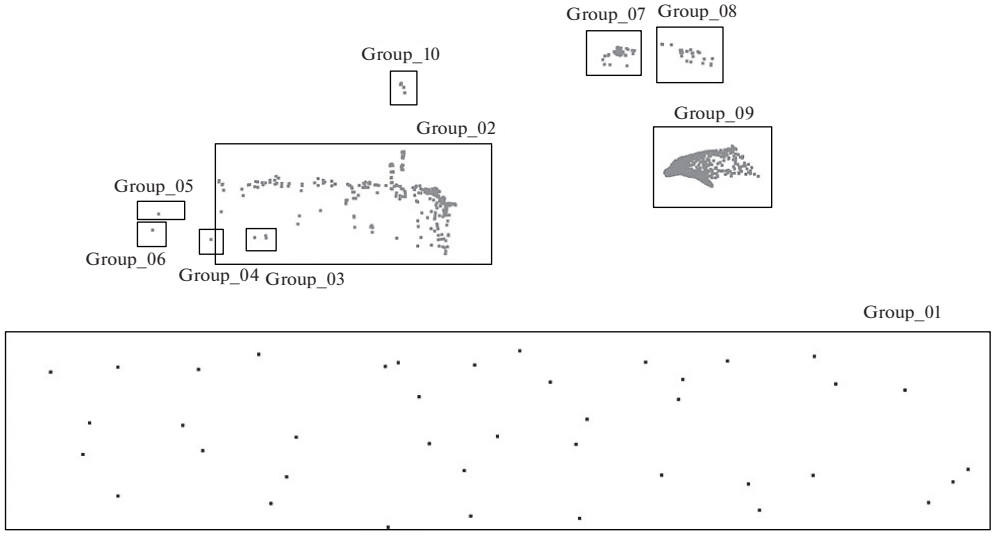


Рис. 9. Показана проекция “вид сбоку”. Каждая из 9 сформированных групп 3D точек выделена прямоугольником.

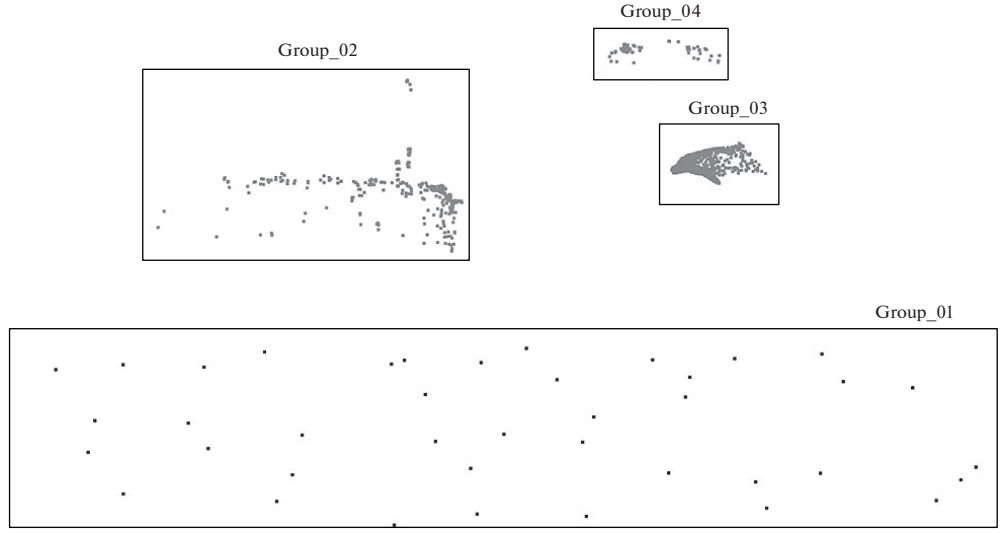


Рис. 10. Выделены 3 группы точек, относящихся к ДО и группа точек, представляющих рельеф дна (после работы алгоритма объединения).

6. СХЕМА ОБРАБОТКИ ДАННЫХ В ЗАДАЧЕ ИДЕНТИФИКАЦИИ И ВОССТАНОВЛЕНИЯ ДВИЖЕНИЯ ДИНАМИЧЕСКИХ ОБЪЕКТОВ ПО ИЗОБРАЖЕНИЯМ

Обработка данных с вычислением траектории движения камеры/робота и ДО осуществляется пошагово. На каждом шаге (в текущей позиции) используется метод визуальной одометрии и выполняется формирование и идентификация точечных моделей статических и динамических объектов с помощью вышеописанной алгоритмической базы.

Вычислительная схема обработки данных на текущем шаге траектории, согласно предлагаемому подходу, состоит из выполнения следующих этапов (рис. 7):

1. Прослеживание с помощью трекера KLT 2D множества особенностей G_{i-1}, сформированного в позиции $(i-1)$ на предыдущем шаге. Построение соответствующего 3D множества $M^{i}_{i-1,i}$, сопоставленного с M^{i-1}.

2. Формирование на стереопаре изображений множества 2D особенностей G_i и построение соответствующего 3D облака точек M^i.

3. Формирование “дальномерного” плотного множества 3D точек R_i.

4. Формирование на объединенном множестве $\{M^{i}_{i-1,i} + R_i + M^i\}$ связных групп точек затравочным алгоритмом (на критерии связности). Группы объединяют точки видимых поверхностей разных объектов или разных частей одного и того же объекта.

Отметим, что в объединенном множестве каждая точка сохраняет статус своей принадлежности к одному из трех указанных множеств, т.к. это необходимо для работы алгоритмов идентификации ДО и вычисления матриц их движения.

5. Объединение в одну группу тех групп точек, которые принадлежат одному и тому же объекту. Объединение выполняется алгоритмом, оценивающим сходство движения.

6. Идентификация группы точек статических объектов. Вычисление методом визуальной одометрии матрицы локального перемещения камеры по сопоставленным точкам статической части сцены (принадлежащим множествам M^{i-1} и $M^{i}_{i-1,i}$).

Исключая из всех полученных групп группу “статика”, получаем ДГ, т.е. группы точек, принадлежащие ДО (недеформируемым и деформируемым).

7. Идентификация ДГ по отношению к ДГ предыдущей позиции.

8. Разделение деформируемых и недеформируемых ДО.

9. Вычисление собственного движения ДО.

Обработка первых двух позиций траектории отличается от приведенной схемы тем, что для позиции 1 отсутствует предыдущий шаг, и, соответственно сегментация точек по группам выполняется на множествах $R_1 + M^{1}_{1,2}$.

7. ВЫЧИСЛИТЕЛЬНЫЕ ЭКСПЕРИМЕНТЫ

Тестирование описанных алгоритмов осуществлялось на подводных виртуальных сценах, в которых съемка ведется с автономного подводного робота (АПР). Пример одной из сцен показан

Таблица 1. Оценка эффективности алгоритмов идентификации ДО

Объект	Кол-во кадров, на которых был определен объект	Усредненная локальная погрешность между соседними кадрами, мм	Накопленная погрешность, мм
подводный аппарат	55	0.4	12.6
батискаф	26	0.6	7.2
дельфин	64	0.6	15.3

Таблица 2. Оценка эффективности всей вычислительной схемы с учетом этапа сопоставления особенностей на стереопаре снимков

Объект	Кол-во кадров, на которых был определен объект	Усредненная локальная ошибка между соседними кадрами, мм	Накопленная ошибка, мм
подводный аппарат	55	0.7	40.4
батискаф	26	0.7	24.8
дельфин	64	0.8	53.1

на рис. 8. В сцене присутствуют три ДО (“подводный аппарат”, “батискаф” и “дельфин”). Алгоритм формирования групп точек по критерию связности сформировал 9 групп 3D точек для данного вида (рис. 9). После работы алгоритма объединения групп точек со сходным движением сформированы 3 группы точек, принадлежащие 3-м указанным динамическим объектам и группа точек, относящаяся к рельефу дна (рис. 10). Данные объекты были прослежены с шагом = 1 кадр на сцене протяженностью в 120 кадров, длительностью 12 сек. Робот/камера и все ДО двигались по прямолинейным траекториям с разными скоростями под разными углами. Время наблюдения объектов (пребывания их в сцене) составило от 2.6 сек (26 кадров) до 6.4 сек (64 кадра).

Другая серия экспериментов была выполнена для оценки точности локализации ДО. Для некоторых позиций камеры (т.е. для отдельных моментов времени) вычислялась локальная (относительная) точность для каждого ДО (погрешность определения координат за время прохождения между двумя соседними позициями) и абсолютная точность (погрешность определения координат, накопленная к данному моменту времени в некоторой внешней системе координат за время движения от начала траектории). Эксперименты выполнялись в два этапа. На первом этапе оценивалась эффективность разработанных алгоритмов при условии безошибочной работы детектора и трекера по сопоставлению особенностей на снимках (использовалась имитация работы детектора и SURF и трекера KLT). Эти результаты представлены в табл. 1.

На втором этапе оценивалась эффективность всей вычислительной схемы, в том числе, с учетом ошибок, вносимых на этапе обработки стереопар снимков (табл. 2).

Как видно из таблиц, полученные результаты подтвердили эффективность предложенных алгоритмов и вычислительной схемы в целом. Этот результат соответствует ранее полученным авторами в [1] оценкам точности локализации.

8. ЗАКЛЮЧЕНИЕ

Рассмотрена модификация ранее предложенного авторами подхода к решению 3D SLAM задачи для динамических сцен, основанного на точечном представлении объектов. Модификация заключается в реализации более эффективной алгоритмической основы, чем это было в начальной версии, а также в реализованных возможностях генерации расширенного точечного представления (за счет применения “виртуального дальномера”) и идентификации динамических объектов с неизменяемой и изменяемой формой.

Расширенное точечное представление позволяет получать более плотные точечные модели ДО. Результаты вычислительных экспериментов на модельных сценах подтвердили правильность выбранных решений. Дальнейшее развитие работы предполагает: совершенствование алгоритмической базы; тестирование программ на разных наборах данных; решение задачи 3D реконструкции ДО с использованием вычисляемых траекторий объектов.

9. БЛАГОДАРНОСТИ

Работа выполнена при частичной финансовой поддержке РФФИ (проект № 18-07-00165).

СПИСОК ЛИТЕРАТУРЫ

1. *Бобков В.А., Кудряшов А.П., Мельман С.В.* О восстановлении движения динамических объектов по стереоизображениям // Программирование. 2018. № 3. С. 29–42 (A. Bobkov, A. P. Kudryashov, S. V. Mel'man. On the Recovery of Motion of Dynamic Objects from Stereo Images // Programming and Computer Software, 2018, Volume 44, Issue 3, pp. 148–158).
2. *Hasler N., Rosenhahn B., Thormahlen T., Wand M., Gall J., Seidel H.P.* Markerless motion capture with unsynchronized moving cameras // Conference on Computer Vision and Pattern Recognition, 2009, pp. 224–231.
3. *Ballan L., Brostow G.J., Puwein J., Pollefeys M.* Unstructured video-based rendering: Interactive exploration of casually captured videos. ACM Transactions on Graphics // Proceedings of SIGGRAPH, 2010. P. 134–146.
4. *Taneja A., Ballan L., Pollefeys M.* Modeling dynamic scenes recorded with freely moving cameras // Conference on Computer Vision, 2011. P. 613–626.
5. *Wedel A., Brox T., Vaudrey T., Rabe C., Franke U., Cremers D.* Stereoscopic scene flow computation for 3D motion understanding // Intl. J. of Computer Vision. 2011. V. 35. № 1. P. 29–51.
6. *Alcantarilla P., Yebes J., Almazn J., Bergasa L.* On combining visual SLAM and dense scene flow to increase the robustness of localization and mapping in dynamic environments // International Conference on Robotics and Automation, May 2012. P. 1290–1297.
7. *Keller M., Lefloch D., Lambers M., Izadi S., Weyrich T., Kolb A.* Real-time 3d reconstruction in dynamic scenes using point-based fusion // Proc. of Joint 3DIM/3DPVT Conference (3DV). 2013. P. 1–8.
8. *Mustafa A., Kim H., Guillemaut J.-Y., Hilton A.* General Dynamic Scene Reconstruction from Multiple View Video // International Conference on Computer Vision. 2015. P. 900–908.
9. *Mustafa A., Kim H., Guillemaut J.-Y., Hilton A.* Temporally coherent 4D reconstruction of complex dynamic scenes // IEEE Conference on Computer Vision and Pattern recognition. 2016. P. 223–245.
10. *Lefloch D., Kluge M., Sarbolandi H., Weyrich T., Kolb A.* Comprehensive Use of Curvature For Robust And Accurate Online Surface Reconstruction // IEEE Transactions on Pattern Analysis and Machine Intelligence, 2017, Vol. 39, №12, p. 2349–2365.
11. *Камаев А.Н., Сухенко В.А., Карманов Д.А.* Построение и визуализация трехмерных моделей морского дна для тестирования систем технического зрения АНПА // Программирование. 2017. № 3. С. 69–82.
12. *Золотов В.А., Петрищев К.С., Семенов В.А.* Исследование методов пространственного индексирования динамических сцен на основе регулярных октодеревьев // Программирование. 2016. № 6. С. 59–66.
13. *Bobkov V.A., Ronshin Y.I.* GPU Implementation of Depth Map Algorithm // The First Russia and Pacific Conference on Computer Technology and Applications, Vladivostok, 6–9 September, 2010, p. 382–387.
14. *Бобков В.А., Роньшин Ю.И., Кудряшов А.П., Машенцев В.Ю.* 3D SLAM по стереоизображениям // Программирование. 2014. № 4. С. 5–12. (V. A. Bobkov, Yu. I. Ron'shin, A. P. Kudryashov, and V. Yu. Mashentsev. 3D SLAM from Stereoimages // Programming and Computer Software, 2014. Vol. 40, No. 4, 2014, pp. 159–165).
15. *Bobkov V.A., Mel'man S.V., Kudryashov A.P.* Fast computation of local displacement by stereo pairs // Pattern Recognition and Image Analysis, 2017. V. 27. № 3. P. 458–465.
16. *Bobkov V.A., Kudryashov A.P., Mel'man S.V., Shcherbatyuk A.F.* Autonomous Underwater Navigation with 3D Environment Modeling Using Stereo Images // Gyroscopy and Navigation, 2018. V. 9. № 1. P. 67–75.
17. *Bay H., Ess A., Tuytelaars T., Van Gool L.* Speeded-Up Robust Features (SURF) // Computer Vision and Image Understanding, 2008. V. 110. P. 346–359.
18. *Lucas B.D., Kanade T.* An Iterative Image Registration Technique with an Application to Stereo Vision // International Joint Conference on Artificial Intelligence, 1981. P. 674–679.

ПРОГРАММИРОВАНИЕ, 2020, № 1, с. 15–28

ИНФОРМАЦИОННАЯ БЕЗОПАСНОСТЬ

УДК 004.56+004.93

ОСОБЕННОСТИ НЕПРЕРЫВНОЙ ИДЕНТИФИКАЦИИ ПОЛЬЗОВАТЕЛЕЙ НА ОСНОВЕ СВОБОДНЫХ ТЕКСТОВ В РЕЖИМЕ СКРЫТОГО МОНИТОРИНГА

© 2020 г. Е. А. Кочегурова[a],*, Ю. А. Мартынова[a],**

[a] *Национальный исследовательский Томский политехнический университет*
634050 Томск, пр. Ленина, 30, Россия
**E-mail: kocheg@mail.r*
***E-mail: martynova@tpu.ru*

Поступила в редакцию 03.06.2019 г.
После доработки 17.06.2019 г.
Принята к публикации 26.06.2019 г.

Данная работа посвящена исследованию особенностей динамической (непрерывной) идентификации пользователей на основе показателей клавиатурного почерка (КП). Клавиатурная динамика отслеживается в режиме скрытого мониторинга при создании пользователем свободного текста в любом приложении. Анализ подходов к статической идентификации не выявил существенных ограничений на их применение к непрерывной идентификации. Главная особенность непрерывной идентификации состоит в способе сбора динамической информации о клавиатурных нажатиях и коррекции шаблонов зарегистрированных пользователей. Показана эффективность включения в алгоритмы распознавания дополнительных классификационных признаков, например, связанных с частотностью использования букв в текстах. Разработано программное приложение для сбора и анализа образцов КП. Исследования, проведенные в домене пользователей с хорошими навыками работы с компьютером, показали вполне удовлетворительную точность распознавания пользователей – в среднем 87%. Причем, точность не зависит от выбранного метрического расстояния при распознавании и несколько повышается при использовании масштабирующих коэффициентов учета частотности букв алфавита.

DOI: 10.31857/S0132347420010033

1. ВВЕДЕНИЕ

Развитие цифровых и сетевых технологий за последнее десятилетие способствовало расширению возможностей для доступа и хранения конфиденциальной информации на цифровых устройствах. Поэтому и защита информации от несанкционированного доступа становится все более актуальной.

По данным аналитического центра InfoWatch число утечек информации в России возрастает год от года. По сравнению с 2016 г. рост числа утечек в 2017 г. резко возрос, почти на 37%, а в 2018 г. по всему миру на 30% больше утечек конфиденциальной информации. Краже чаще всего подвержены персональные и платежные данные и составляют от всего объема утечек за 2017 г. 64.1% и 21.1% соответственно. По-прежнему главным каналом утечек остается Сеть (броузер, cloud) – 69.8% (2016 + 11.6%). Основными виновными утечки информации оказались сотрудники и руководство организаций – 64% и 50% случаев.

В связи с этим повышается необходимость динамического или непрерывного распознавания пользователей, имеющих доступ к общественным и личным информационным ресурсам.

Кроме технических и организационных мер для защиты информации широко используются средства аутентификации (авторизации) и идентификации пользователей. Наряду с парольными и имущественными методами аутентификации, нередко используются биометрические характеристики. В соответствии с Российскими и международными стандартами (ГОСТ Р 54412-2011 и ISO/IEC 24745), биометрия – наука, которая изучает методы определения идентичности человека на основе физиологических и поведенческих особенностей. Биометрический признак (характеристика) – это уникальная, измеримая характеристика, используемая для верификации отдельно взятого человека [1–3]. Выделяют физиологические и поведенческие характеристики. К первой группе относятся уникальные признаки, полученные человеком при рождении. Например, ДНК, отпечатки пальцев, радужная оболочка гла-

за, рисунок вен, форма ушей и др. Поведенческие же характеристики приобретаются со временем и способны меняться с возрастом или в результате внешнего воздействия. В качестве примера можно назвать голос, рукописный и клавиатурный почерк, походку.

Установлено, что динамика нажатия клавиш или клавиатурный почерк (КП) определяет индивидуальный ритм набора текста и может использоваться как биометрическое средство идентификации личности [4–6]. Отмечено, что динамика нажатия клавиш “не то, что вы набираете, а то, как вы печатаете” [Monrose-1997]. Как поведенческая характеристика, КП по своей природе являются динамической. Обычно он формируется через 6 месяцев работы с компьютером [7]. Поведенческие характеристики включают условно постоянную компоненту, обусловленную физиологией пользователя (его способностями и навыками работы с клавиатурой) и случайную компоненту, определяемую психоэмоциональным состоянием человека. Поведенческие характеристики КП в отличие от физиологических сложнее распознать с высокой точностью, но вместе с тем, сложнее и подделать [4]. Основной акцент данного исследования сделан на анализ особенностей динамического распознавания КП.

Статья структурирована следующим образом. В разделах 2–3 рассмотрены существующие подходы и тенденции клавиатурной динамики. Раздел 4 посвящен особенностям сбора данных и показателям КП. В разделах 5–6 обсуждаются показатели эффективности, алгоритмы и методы клавиатурной идентификации. И, наконец, раздел 7 описывает проведенный эксперимент и его результаты.

2. ПОДХОДЫ К КЛАВИАТУРНОЙ ДИНАМИКЕ

Внедрение системы аутентификации по КП не требует значительных материальных вложений. Единственное необходимое оборудование – это стандартная клавиатура, которой оснащены все компьютеры, и высокоэффективное программное обеспечение.

Первые работы по анализу динамики клавиатурных нажатий относятся к 80-м годам прошлого века. В то время была обнаружена связь пользователя и ритма клавиатурных нажатий; были введены первые биометрические характеристики, способы их получения и исследованы статистические методы распознавания пользователей.

Существует несколько классификаций для клавиатурной идентификации пользователей. Одна из них – по типу создаваемого текста.

I. Статическая аутентификация [8–10]. Проверка пользователя осуществляется во время первичной аутентификации на основе структурированного/предопределенного текста. Предопределенный текст обеспечивает более надежную проверку пользователя, чем только ID/пароль при первичной аутентификации. Также предопределенный текст может дополнять проверку ID/пароль в случае возникающих подозрений.

II. Динамическая или непрерывная аутентификация [10–16]. Непрерывная проверка осуществляется на протяжении всего сеанса работы пользователя на основе произвольного текста, вводимого им в любом программном приложении. Такой скрытый мониторинг обеспечивает дополнительную меру безопасности после авторизации пользователя в системе. Свободный текст в большей степени, чем предопределенный, напоминает реальную среду работы пользователя и позволяет выявить поведенческие характеристики.

Хотя аутентификация на основе пароля проста в разработке и обслуживании, но она оказывается бесполезной в случае выявления пароля злоумышленниками. Время набора пароля часто бывает недостаточным, чтобы оценить изменение динамики нажатия клавиш. И если статический режим используется преимущественно при первичной аутентификации, то основной целью скрытого мониторинга является установление подлинности пользователя или его психоэмоционального состояния в процессе работы в корпоративной системе.

Одним из недостатков клавиатурной идентификации является невысокая точность по сравнению с другими биометрическими системами. Поэтому повышение качества распознавания попрежнему является актуальной задачей, особенно для динамической.

Целью данной работы является исследование возможности применения подходов статической идентификации для динамических данных и динамической идентификации.

При автоматической идентификации пользователей на основе КП существует ряд особенностей в организации получения и распознавания входных данных. Основные из них следующие:

– сбор статистики клавиатурных нажатий;

– выбор временных показателей клавиатурной динамики;

– создание шаблонов (профилей) пользователей;

– выбор показателей эффективности распознавания;

– алгоритмы и методы распознавания.

3. СОСТОЯНИЕ ВОПРОСА И АКТУАЛЬНОСТЬ КЛАВИАТУРНОЙ ИДЕНТИФИКАЦИИ

Клавиатурная динамика в последние годы является областью активных исследований и ис-

пользования в системах информационной безопасности вследствие низкой стоимости и простоты сопряжения с существующими системами. Впервые работы по использованию клавиатурных нажатий, как средства идентификации пользователей, были связаны с измерением (изучением) моторных навыков нажатия клавиш. И только в 90-е годы [17] клавиатурную динамику стали рассматривать, как поведенческую характеристику почерка.

Первые обзоры исследований КП [17, 18] относятся к началу 2000 годов. В частности, в работах 2004–2010 гг. [1, 4, 11, 19, 20] были проанализированы актуальные классификаторы на основе статистических методов и нейронных сетей. Было установлено, что вероятностные методы более привлекательны в вычислительном плане, но имеют более низкую точность аутентификации. Появились рекомендации по созданию публичных наборов данных для аутентификации. Также в обзорах тех лет отмечено, что большинство работ относится к статической аутентификации пользователей, т.е. на основе пароля или фиксированного текста. Большинство исследователей подтвердили, что системы аутентификации на основе клавиатурной динамики имеют потенциал в области кибербезопасности и биометрического мониторинга.

В обзорах последних лет 2011–2018 [5, 9, 16, 21–27] кроме исследования развития методов классификации и распознавания пользователей, проанализированы факторы, влияющие на производительность систем аутентификации. В эти годы помимо использования стандартных клавиатур персонального компьютера, активно развивается аутентификация пользователей сенсорных клавиатур, мобильных устройств [28–31], и на основе web-сервисов [32]. Количество публикаций по клавиатурной биометрике достаточно быстро увеличивается, особенно в последнее десятилетие. На рис. 1 показано число публикаций и тенденции роста: в период 1998–2012 данные взяты из [23] и [9], 2013–2017 гг. данные получены нами на основе анализа баз данных (БД) Scopus, IEEEXplore, Science Direct, электронной библиотеки ACM. В [9] изложена технология выявления прямых ссылок на наиболее значимые результаты в этой области. И из более, чем 2000 тематических ссылок на клавиатурную динамику выделены 16 прямых ссылок на работы, опубликованные в 2004–2009 гг. Именно в эти годы были заложены теоретические основы клавиатурной динамики. Этот вывод подтверждают и анализ опубликованных патентов [3].

Исследования нынешнего десятилетия значительно расширили прикладной характер клавиатурной биометрии. Кроме аутентификации на базе мобильных устройств, web-приложений, появились работы по гендерной и возрастной

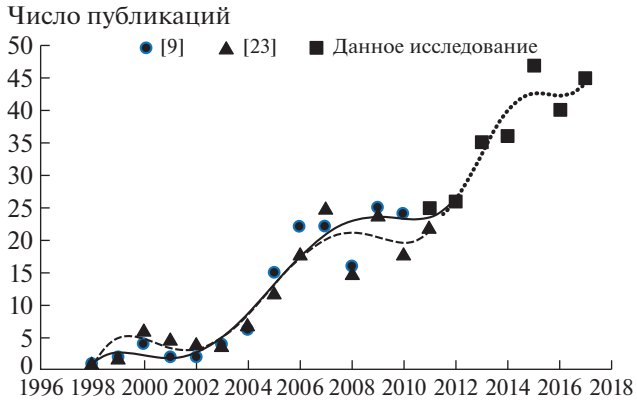


Рис. 1. Тенденции роста числа публикации по клавиатурной динамике.

идентификации на базе клавиатурной динамики [33]. Перспективна линия работ, связанная со смешанной аутентификацией [24, 34]. Например, на базе КП и особенностей лица [34]. Как и динамическая идентификация КП более перспективна, чем статическая, при решении современных задач биометрического распознавания: гендерного, психоэмоционального состояния.

Необходимо отметить, что клавиатурное распознавание не может быть единственным средством биометрической аутентификации, как и не может полностью заменить парольную систему аутентификации. Однако несомненные достоинства, такие как в скрытый мониторинг, низкая стоимость внедрения, высокая степень идентификации и простота интеграции с другими системами безопасности, делают клавиатурную аутентификацию достойным дополнительным средством безопасности.

В реализации системы клавиатурного распознавания участвуют три подсистемы:

а) сбор данных и извлечение характеристик КП;

б) формирование клавиатурного профиля;

в) идентификация пользователя.

4. СБОР ДАННЫХ И ХАРАКТЕРИСТИКИ НАЖАТИЯ КЛАВИШ

Сбор данных о динамике нажатия клавиш – это начальный этап реализации клавиатурной идентификации. Его эффективность полностью определена специальным программным обеспечением и не требует дополнительного оборудования. При клавиатурных нажатиях операционная система фиксирует код ANSI, связанный с нажатой клавишей, время ее нажатия и отпуска. Как событие нажатие клавиши можно измерить с точностью до миллисекунды. На этом и основан сбор первичных данных по динамике нажатия клавиш пользователями.

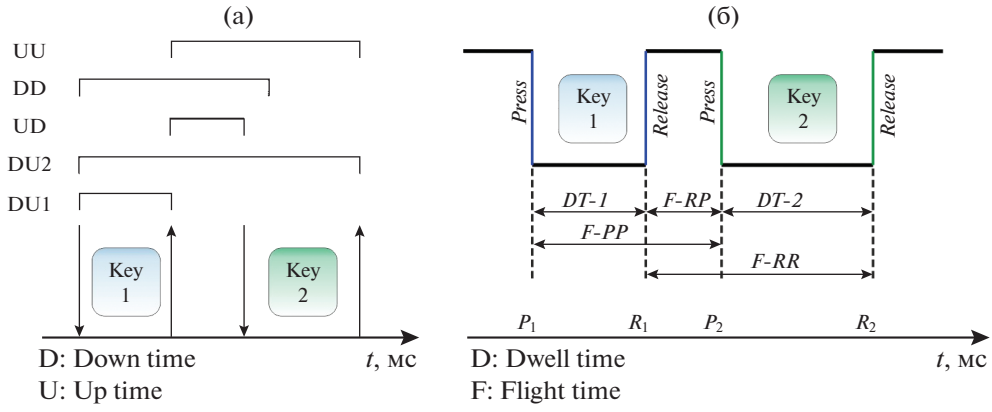


Рис. 2. Характеристики клавиатурного почерка.

В целом динамика нажатия клавиш определяет процесс оценки ритма набора текста и формирует собственный шаблон (профиль, образ) отдельного человека. Шаблон пользователя, как и его рукописная подпись или текст, создается под действием определенных нейрофизиологических факторов. Это делает шаблоны КП уникальными [9]. При использовании клавиатуры компьютера у человека задействовано до 140 мышц [36, 37], 20% от которых управляют нажатием клавиш. Соответственно 28-мерная задача управления позволяет говорить об уникальности КП-пользователей и его потенциальной значимости для распознавания личности.

Шаблон нажатия клавиши способен обеспечить уникальную функцию аутентификации. Он основан на достаточно большом наборе показателей, извлекаемых из статистики о клавиатурных нажатиях, которые характеризуют длительность, задержку нажатия клавиш и скорость набора текста. Большинство исследователей [9, 10, 23, 35] используют следующие показатели:

– время удержания клавиши;

– паузы между нажатиями;

– число ошибок при вводе;

– степень аритмичности при наборе;

– скорость набора;

– особенности использования служебных клавиш.

В некоторых предыдущих исследованиях также учитывалось использование давления на клавиши [24], однако для этого требуется специализированная клавиатура. Но ввиду повсеместного использования устройств с сенсорным экраном интерес к таким работам повышается. При этом часть характеристик: аритмичность, скорость набора, исправление ошибок чаще используются, как дополнительные к основным временным параметрам.

Впервые обоснование для использования временного шаблона нажатий клавиш было выполнено еще в 1980 г. [38]. И на протяжении десятилетий актуальность использования шаблона остается по-прежнему высокой. Для формирования шаблона существует необходимость извлекать эффективные характеристики из зафиксированных операционной системой нажатий.

Большая часть исследователей КП анализируют в своих работах [22, 9, 35, 10, 39] временные характеристики между двумя последовательными нажатиями клавиш, так называемые диграфы (Di-graph). Существует два основных диграфа: время ожидания (Dwell time DT, Hold time HD) и время задержки (Flight time FT или keystroke latency). Времена ожидания и задержки имеют наглядную геометрическую иллюстрацию. На рисунках 2а, 2б приведены эти характеристики в двух основных нотациях графического изображения временных характеристик.

На рисунке 2-а) представлены следующие характеристики клавиатурной динамики в терминах: Down/UP :

DU1: временной интервал между моментами, в которые нажата и отпущена клавиша, т.е. время удержания клавиши (Dwell time или Hold time).

DU2: временной интервал между моментами нажатия одной клавиши и отпусканием следующей клавиши.

UD: временной интервал между моментами, в которых одна клавиша отпущена, а другая нажата, т.е. пауза. (Flight time).

DD: временной интервал между моментами нажатия двух клавиш.

UU: временной интервал между моментами отпуска двух клавиш.

На рисунке 2б в терминах Press/ Release приведены следующие времена задержек:

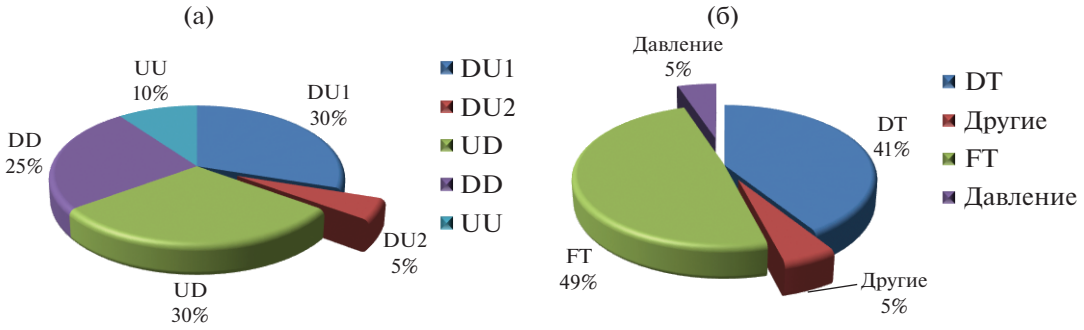


Рис. 3. Процентное распределение временных признаков.

F-RP=UD;

F-PP=DD;

F-RR=UU.

DU – важнейшая характеристика, под которой понимается промежуток времени, в течение которого клавиша находится в нажатом состоянии. Как правило, диапазон изменения удержания клавиши (15–150) миллисекунд.

Манеру создания текстов во многом определяет число перекрытий (наложений) нажатия клавиш. Наложение происходит тогда, когда одна клавиша еще не отпущена, а другая уже нажата. С повышением скорости набора текста увеличивается число наложений. В случае наложений пауза UD принимает отрицательное значение.

4.1. Функции динамики нажатия клавиш

Хотя большинство исследователей при анализе динамики КП считают важнейшими характеристиками время удержания клавиши (DT/DU) и ожидания нажатия (UD), определенный вклад в профиль пользователя вносит анализ задержек (keystroke latency) [11, 23, 45]. Кроме указанных выше диграфов существуют аналогичные характеристики для трех клавиш – триграммы. И они показали неплохие результаты для аутентификации [4]. В обзорах [9, 22, 23] приведен анализ частоты использования временных меток разными исследователями. На рисунках 3 приведены диаграммы процентного соотношения основных временных меток.

Общее число исследователей для представленных данных на рис. 3а и 3б различно (40 [9] и 187 [23]), но общая закономерность отчетливо прослеживается. Основными временными характеристиками клавиатурной динамики являются удержания клавиши (DT/DU) и обобщенная группа времени задержек (FT), которая на рис. 3-а) разбита на 3 части (UD + DD + UU).

Выбранные показатели о клавиатурных нажатиях пользователя собираются, обрабатываются и на этой основе формируется эталонный профиль пользователя. Большинство исследований в области анализа нажатия клавиш собирают данные из структурированного и предопределенного текста.

При непрерывном (динамическом) сборе данных необходимо возникает ряд дополнительных вопросов [6]:

– объем данных для формирования шаблонов. Условно объемы данных при непрерывном исследовании клавиатурной динамики принято разделять на небольшой (менее 1000 нажатий), средний (менее 6000 нажатий) и большой (более 6000 нажатий) на каждого пользователя [14, 23, 24]. Иногда в свободных текстах анализируется количество слов. По данным [23] 57% исследователей анализируют короткие тексты, 24% – длинные, оставшиеся 19% распределены между текстами, содержащие только цифры и тексты неизвестного содержания;

– формирование и использование общедоступных БД профилей пользователей. Первые БД клавиатурных нажатий появились в 2009 года. БД содержат разное количество пользователей (30–300) и разное число повторений в сессию. Но абсолютное большинство БД содержат клавиатурные данные, собранные на основании паролей и фиксированных текстов и, следовательно, малопригодны для скрытого и непрерывного мониторинга;

– полное отсутствие подобных БД с шаблонами пользователей на русском языке. А это приводит к необходимости сбора данных каждым исследователем. И соответственно развитие теории распознавания клавиатурной динамики и затрудняет сопоставление результатов в равных условиях.

5. ОЦЕНКА ЭФФЕКТИВНОСТИ АУТЕНТИФИКАЦИИ

Эффективность идентификации показывает способность метода отличить подлинного пользователя домена и предотвратить несанкционированный доступ.

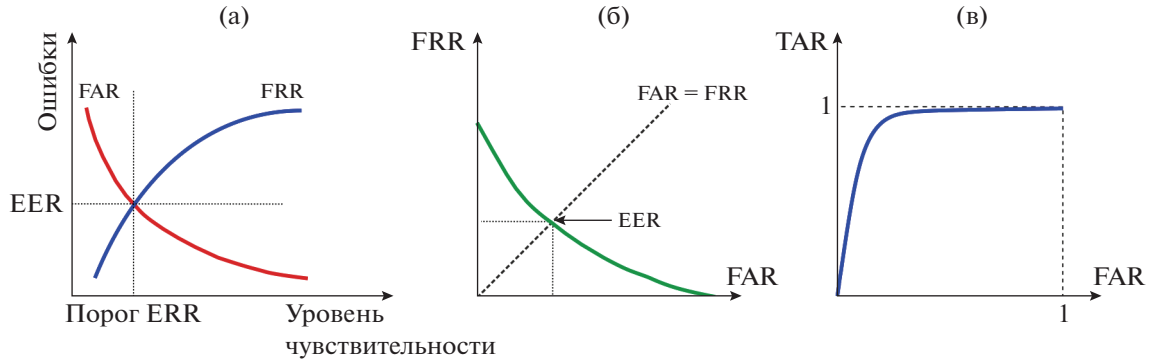


Рис. 4. Показатели эффективности клавиатурной идентификации.

При сравнении двух образцов КП возможны следующие варианты развития событий [40, 41]:

True Accept (TA): образцы принадлежат одному и тому же пользователю, и система определяет образцы как схожие – это ожидаемое событие;

True Reject (TR): образцы принадлежат разным пользователям, и система определяет их как не схожие – также ожидаемое событие;

False Reject (FR): образцы принадлежат одному и тому же пользователю, но система определяет их как несхожие – т.е. опровергается верная гипотеза;

False Accept (FA): образцы принадлежат разным пользователям, но система определяет их как схожие – принимается ложная гипотеза.

Показатели эффективности, используемые в разных клавиатурных исследованиях, можно обобщить в четыре основные показателя частоты ошибок.

False Rejection Rate (FRR) – оценка ложного отклонения, известна в статистической радиотехнике, как ошибка I рода. FRR определяет процент случаев, когда законный пользователь ошибочно отклоняется. Вычисляется, как отношение ложно отклоненных пользователей к общему количеству пользователей системы.

$$\text{FRR} = \frac{\text{Количество ложных отказов}}{\text{Общее количество попыток}} \quad (1)$$

Более низкая FRR подразумевает меньшее отклонение пользователей и более легкий доступ для них.

False Acceptance Rate (FAR) – ошибка ложного принятия. FAR или ошибка II рода определяет процент случаев принятия нелегальных пользователей. Вычисляется, как отношение ложно принятых неавторизованных пользователей к общему количеству пользователей системы.

$$\text{FAR} = \frac{\text{Количество ложных совпадений}}{\text{Общее количество попыток}} \quad (2)$$

Гипотетически ошибки FRR и FAR варьируются в зависимости от уровня чувствительности алгоритма (порогового значения) как показано на рис. 4а: когда одна ошибка уменьшается, другая увеличивается.

Более высокие значения FAR обычно предпочтительнее в системах, где безопасность не имеет первостепенной важности, тогда как более высокие значения FRR являются предпочтительным в приложениях с высокой степенью защиты [1]. Компромисс между FAR и FRR должен определяться целями конкретной прикладной задачи. Например, низкий пропуск нелегальных пользователей (FAR) соответствует высокому пороговому значению (чувствительности), но приводит к большому отклонению зарегистрированных пользователей (FRR), т.е. к их низкому пропуску.

Между ошибками FAR и FRR существует компромисс, т.е. можно уменьшить одну погрешность за счет другой, регулируя порог принятия решения. На рисунке 4б представлена кривая DET (Detection Error Trade-off) [41, 42], позволяющая определить компромисс между ошибками I рода (FRR) и II рода (FAR). По ней можно установить одну из желаемых ошибок и увидеть жертву другой.

Часто используемая Equal Error Rate (EER) – равная частота ошибок – для определения общей точности системы распознавания. EER представляет значение ошибки, когда FAR и FRR принимают равные значения [10]. В отличие от FAR и FRR, ошибка EER не зависит от уровня чувствительности алгоритма аутентификации. Чем ниже значение EER, тем эффективнее система распознавания при заданном пороговом значении. Независимый от порога EER более подходит для оценки эффективности алгоритмов распознавания.

Показатели FAR, FRR и EER весьма популярны и перспективны в системах клавиатурной аутентификации. Менее часто используемая характеристика ROC (Receiver Operating Characteris-

Рис. 5. Частота использования методов клавиатурного распознавания.

tic) [41, 42], позволяет увидеть предельные значения показателей эффективности. Характеристика ROC отображает компромисс между верным (TAR) и ошибочным (FAR) пропуском пользователя при различных пороговых значениях. Верхний левый угол графика представляет собой идеальную точку, где TAR равно единице, а FAR равно нулю.

6. АЛГОРИТМЫ РАСПОЗНАВАНИЯ ПОЛЬЗОВАТЕЛЕЙ

В разделе 2 данной работы описано состояние вопроса, актуальность и подходы к клавиатурной идентификации в историческом аспекте. Большинство алгоритмов распознавания клавиатурных профилей, используемых на протяжении трех десятилетий, относятся к статической идентификации пользователей. Обзорные работы [3, 6, 9, 14, 16, 22–24], опубликованные за последние 5 лет, отражают современное состояние клавиатурного распознавания. Практически каждое опубликованное исследование включает подробные таблицы с перечислением авторов, алгоритмов классификации и распознавания, временных показателей и достигнутых показателей эффективности. Резюмируя результаты этих исследований можно отметить:

– количество исследований по динамической аутентификации очевидно мало, не более 10% [23];

– основные исследования по-прежнему посвящены использованию стандартной клавиатуры;

– 80% исследований основаны на коротких или предопределенных текстах.

Подходы к динамической и статической идентификации принципиально не отличаются и с позиции распознавания условно разделены на 3 основные группы:

– оценка метрических расстояний;

– статистические методы;

– методы машинного обучения.

Хотя статистические методы и были первыми в задаче распознавания КП, их использование актуально и сейчас во всех задачах клавиатурной идентификации. Вероятностный подход оценивает вероятность того, что клавиатурный профиль принадлежит пользователю некоторого домена (БД). Используемые методы вероятностного моделирования включают байесовские методы, скрытую Марковскую модель, функцию гауссовой плотности и взвешенную вероятность. Анализируемыми статистическими показателями являются: средние, медианные, стандартные отклонения, статистический t-критерий для оценки подобия [10, 13, 25] и др. Однако разделение на статистические и вероятностные методы условно, и они могут быть объединены в один класс методов. Используя данные [23, 39] по методам распознавания КП на рис. 5 показано процентное распределение методов распознавания и классификации КП.

Из рисунка 5 следует, что оценка расстояния (метрики) является самым популярным методом распознавания и достигает по частоте использования 23%. В рамках этого подхода вычисляется расстояние между эталонным и текущим профилем пользователя, которое затем сопоставляется с пороговым значением. Наиболее часто используемыми метрическими расстояниями являются меры Евклида, Манхэттенская, Хэмминга, Махаланобиса [11, 24, 26]. В том числе на основе относительных расстояний для длинного и свободного текста [14].

Следующим по частоте использования (16%) является метод искусственных нейронных сетей (ИНС), относящийся к категории методов распознавания образов на основе машинного обучения. Предполагается, что ИНС более эффективна, чем статистические методы [15, 24, 43]. Однако сложность использования ИНС, как классификатора КП, состоит в необходимости иметь для обучения

Таблица 1. Распределение букв алфавита по частотным диапазонам

Русский алфавит		Английский алфавит	
Частотный диапазон %	Буквы алфавита	Частотный диапазон %	Буквы алфавита
[10.98–5.47]	о/е/а/и/н/т/с	[12.02–6.02]	e/t/a/o/i/n/s/r
[4.75–1.59]	р/в/л/к/м/д/п/у/я/ы/ь/г/з/б	[5.92–2.09]	h/d/l/u/c/m/f/y/w
[1.45–0.013]	я/й/х/ж/ш/ю/ц/щ/э/ф/ъ/ё	[2.03–0.07]	g/p/b/v/k/x/q/j/z

сети, образцы легальных и нелегальных пользователей. Общие архитектуры ИНС представляют собой многослойный персептрон (MLP) [45], радиальную базовую функциональную сеть (RBFN), квантование векторного обучения (LVQ) и самоорганизующуюся карту (SOM) [43]. Однако, непрерывная идентификация пользователей предполагает обновление шаблонов пользователей, что приводит к переобучению сети и увеличению времени идентификации.

К группе распознавания КП на основе машинного обучения также относятся ряд известных алгоритмов, включая дерево решений, нечеткую логику и эволюционные вычисления.

Дерево принятия решений весьма популярный метод, благодаря своей низкой вычислительной сложности. Но ввиду рекуррентности процедуры распознавания, метод эффективен лишь при небольших множествах легальных и нелегальных пользователей [13, 41].

Нечеткая логика использует многозначную логику для моделирования задач с неоднозначными данными. Основная идея заключается в построении границ области принятия решений на основе данных обучения с функциями принадлежности и нечеткими правилами. После выделения пространства признаков и вычисления значений принадлежности производится идентификация категории, которой принадлежит исследуемый шаблон.

Эволюционные вычислительные методы, основанные на идее естественного отбора, в задаче клавиатурной идентификации включают ряд известных подходов: генетические алгоритмы (GA), алгоритмы роевого интеллекта и ряд других [5]. Данные оптимизационные алгоритмы позволяют осуществить направленный поиск максимального совпадения анализируемых клавиатурных шаблонов, тем самым повышая точность распознавания.

Другой известный классификатор – метод опорных векторов (SVM) [10, 13, 12, 24]. Концепция этого метода заключается в том, чтобы сначала определить, как два класса функций легальных и нелегальных пользователей отличаются друг от друга. Затем создается граница, которая лучше всего разделяет эти классы функций. И по расположению проверяемого шаблона относительно выделенной границы выделяют законных и незаконных пользователей. SVM обладает показателями эффективности распознавания сопоставимыми с нейронной сетью при меньших вычислительных затратах. Однако производительность заметно снижается, когда набор временных показателей велик.

Кластерный анализ при анализе КП – это метод объединения похожих клавиатурных профилей и образцов. Цель состоит в том, чтобы сгруппировать образец с аналогичными клавиатурными признаками для формирования однородного кластера. Профили из разных кластеров очень разнородны, но очень похожи между собой в одном кластере. К этой группе методов относятся K-means, K-star, DBScan, KNN и др. [10, 14, 24, 45].

Разделение методов распознавания КП по группам и подходам весьма условно. Разные подходы могут приводить к одной и той же модели, но при этом методы ее обучения могут быть разными.

7. ОПИСАНИЕ ЭКСПЕРИМЕНТА И РЕЗУЛЬТАТЫ

Описанные выше подходы к статической идентификации не выявили существенных ограничений на их применение к непрерывной (динамической) идентификации. Главная особенность непрерывной идентификации состоит в способе сбора динамической информации о клавиатурных нажатиях и коррекции шаблонов зарегистрированных пользователей. В связи с этим при проведении эксперимента и его обработке были поставлены следующие задачи:

– сбор и актуализация динамических наборов данных;

– расширение известных подходов статической идентификации для случая динамического распознавания личности на основе свободных и длинных текстов;

– повышение эффективности процедуры непрерывной идентификации в пространстве выделенных клавиатурных признаков.

Анализ подходов к статической идентификации и особенностей непрерывной идентификации позволил сформировать методику автоматического распознавания пользователей некоторого домена при создании им произвольных текстов. В основе

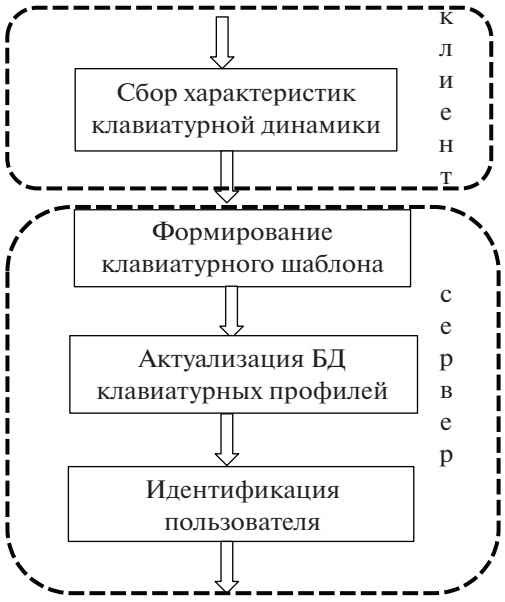


Рис. 6

методики лежит расширение проанализированных подходов к статической идентификации. Основные этапы автоматического распознавания следующие.

I. Непрерывный сбор информации о клавиатурных нажатиях.

Эта информация собирается на основе работы пользователя в любом приложении Windows: в окне браузера, тестовом редакторе и других приложениях. ОС с помощью механизма перехвата сообщений Windows-hook позволяет зафиксировать любое нажатие (отпускание) клавиш функцией события до того, как оно дойдет до приложения [12]. При этом сохраняется информация о том, какая клавиша была нажата или отпущена и время возникновения события клавиатуры.

Для поддержания обычного психоэмоционального состояния пользователь, как правило, не информирован о мониторинге своих действий на клавиатуре. При этом режим скрытого мониторинга не нарушает конфиденциальности персональных данных, т.к. не производит семантический анализ вводимых данных.

II. Актуализация динамических наборов данных.

В процессе работы на клавиатуре и при создании текстов несколько изменяется ритм набора. Поэтому при динамической идентификации личности имеет смысл анализировать не весь поток данных, а определенный объем последних символов. В данной работе минимально допустимый объем свободного текста составляет 200 символов. Далее происходит накопление объема клавиатурных нажатий до 1000 символов и последние 1000 символов постоянно обновляются на протяжении сеанса работы пользователя. Данные объемы статистических данных гарантируют получение несмещенных и состоятельных оценок для временных характеристик КП, а также репрезентативность (частотность) букв алфавита.

III. Формирование временного показателя клавиатурной динамики.

Анализ показателей, характеризующих динамику нажатия клавиш показал, что наиболее часто в прикладных задачах рассматриваются время удержания (DU) и пауза (UD), рис. 3.

В данном исследовании для повышения эффективности сырых данных было принято связать временные характеристики с частотой использования букв алфавита. Частотный диапазон использования букв достаточно широк: [0.013%–10.98%] для букв русского алфавита и [0.07%–12.02%] для английского.

Клавиатурный ритм в значительной мере зависит от частоты появления букв алфавита в тексте. Буквы, редко используемые в текстах, требуют большего внимания и времени при нажатии,

Таблица 2. Характеристики клавиатурного профиля

Клавиша	а	б	о	п	я	а	б	о	п	я
Пользователь	nvb16					ksk12				
Мат. ожидание	76.	85.5	77.5	74.0	81.4	83.9	81.7	67.5	88.8	95.1
Ср. отклонение	2.7	8.3	3.1	5.5	6.1	4.1	17.6	3.8	7.9	9.3
Max	82.3	106	83.2	89.5	95.5	91.8	135	74.6	107	117.3
Min	71.9	60	70.3	65.8	71	77.2	64.4	62.3	73.6	82.2
Пользователь	lrd1					vve15				
Мат. ожидание	99.6	102.8	100.6	100.4	116.5	75.4	58.0	57.5	75.4	87.1
Ср. отклонение	6.7	7.9	6.4	9.1	13.4	8.5	5.0	4.9	8.3	11.4
Max	113.2	117.3	113.72	119.1	143.2	98.4	65	68.3	101	121.8
Min	90.8	89.4	89.4	81.1	90.6	63.5	43.5	50.5	64.2	68.1

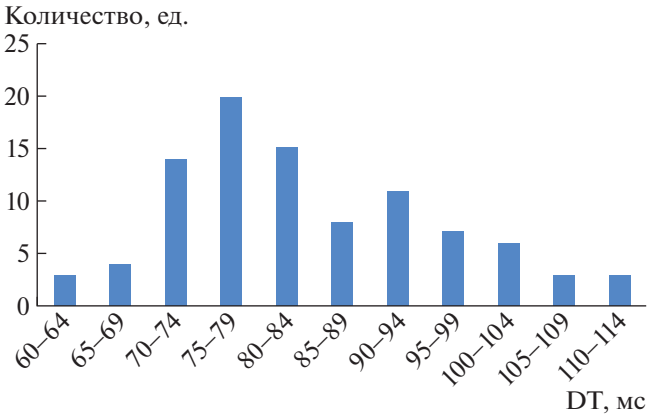


Рис. 7. Распределение времени удержания клавиши, мс.

так как не ассоциированы с мышечной памятью. Поэтому было предложено разделить частотные диапазоны русского/английского алфавита в соотношении 0.5/0.3/0.2. В таблице 1 представлено это разделение. Такие же нормирующие коэффициенты введены для временных характеристик каждой группы букв.

IV. Создание шаблонов (профилей) пользователей.

В БД хранятся клавиатурные шаблоны каждого известного системе пользователя домена, иногда за несколько последних сеансов работы пользователя.

Для каждой клавиши в текущем сеансе пользователя вычисляются средние значения основных временных характеристик клавиатурной динамики. Полученная информация формирует клавиатурный профиль (шаблон, почерк) пользователя и записывается в БД пользователей.

При динамическом распознавании требуются по каждому пользователю обновлять стек сеансовых шаблонов. В работе принято, что, если список шаблонов конкретного пользователя становится слишком длинным (более десяти образцов), самый старый образец удаляется. Так происходит актуализация клавиатурного профиля отдельных пользователей и БД домена.

V. Алгоритмы и методы распознавания.

Проведенные исследования показали, что самым популярным методом распознавания пользователя по его КП по-прежнему является оценка расстояния (метрики) между текущим и эталонным образцами КП. И в соответствии с рисунком 5 частота использования метода в прикладных исследованиях КП составляет 23%. Традиционно используются следующие меры близости: Евклидова, Манхэттенская, Махаланобиса и Хемминга. Евклидова метрика весьма популярна, так как является естественным расстоянием в Евклидовой геометрии. Манхэттенская или метрика городских кварталов, введена Г. Минковским, по сравнению с Евклидовой уменьшает влияние отдельных выбросов, так как не возводит разность в квадрат. Расстояние Махаланобиса применяется в случае ненулевой корреляции переменных и эта метрика инвариантна к масштабу. В данной работе алгоритм распознавания исследован с использованием Евклидовой и Манхэттенской метрики.

Для идентификации пользователя, каждый введенный образец почерка сравнивался с клавиатурными шаблонами из БД. При этом возможны две ситуации.

■ Введенный образец аналогичен одному из имеющихся в БД шаблонов почерка. Образцы считаются аналогичными, если расстояние между векторами характеристик этих образцов не превышают некоторого порогового значения.

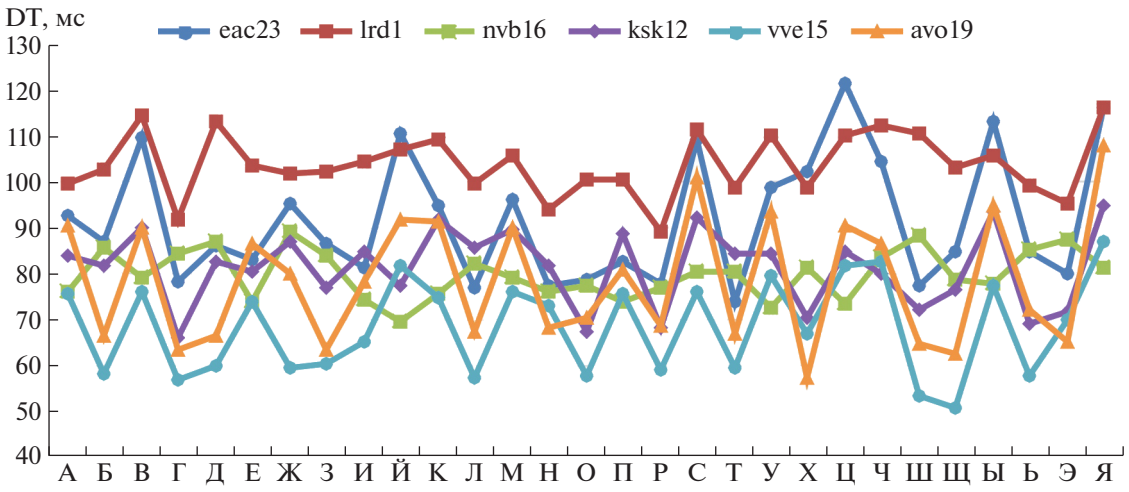


Рис. 8. Среднее время удержания клавиш пользователями домена, мс.

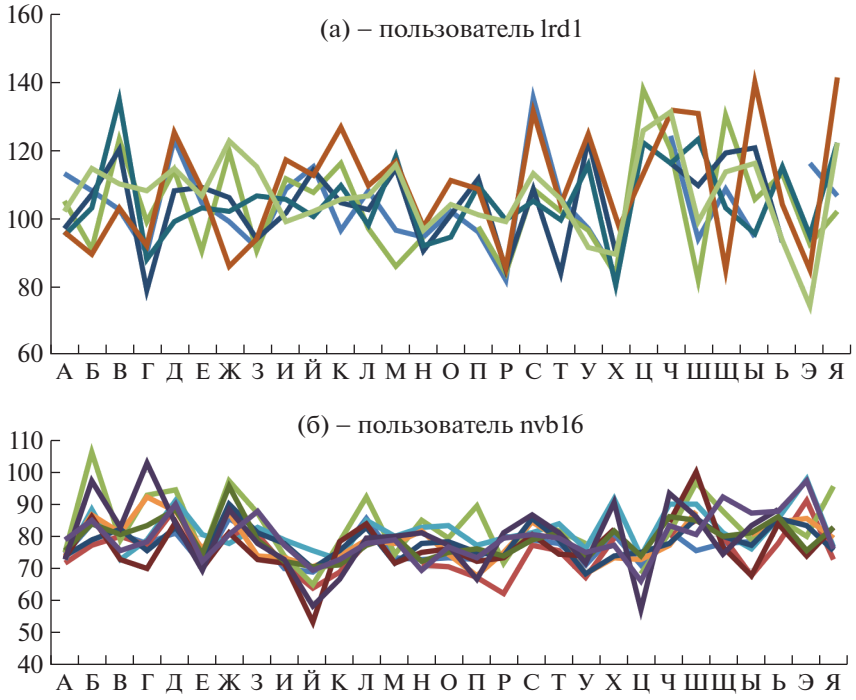


Рис. 9. Среднее время удержания клавиш в разных сеансах, мс.

Иначе образцы – непохожи. И, если образец аналогичен одному из имеющихся, система идентифицирует пользователя и добавляет новый образец почерка в стек сеансов в БД.

■ Новый образец не похож ни на один из имеющихся образцов. В этом случае фиксируется ошибка аутентификации и пользователь идентифицируется, как неопознанный.

Для реализации предложенной методики распознавания пользователя было создано программное приложение и проведен эксперимент. В эксперименте приняли участие в общей сложности тридцать пользователей ПК, использующих стандартную клавиатуру компьютера. Исследование было проведено на основе свободно создаваемых пользователями текстов в режиме скрытого мониторинга.

Пользователи домена имели разные уровни навыков набора текста, которые варьировались между умеренными и очень хорошими, что характерно для студентов и преподавателей университета.

Разработанное программное приложение установлено в корпоративном домене университета. Функционально приложение предназначено для непрерывного сбора информации о клавиатурных нажатиях на русском/английском языках и идентификации санкционированного или несанкционированного пользователя. Основные этапы работы приложения приведены на рис. 6.

После накопления достаточного количества данных, они отправляются на сервер программы для дальнейшей обработки. Для повышения надежности передача данных происходит посредством TCP-сокетов. Серверный компонент вычисляет средние значения основных временных характеристик клавиатурной динамики для каждой клавиши в текущем сеансе пользователя.

Непрерывный сбор и анализ клавиатурных данных позволил оценить идентификационные возможности пользовательских шаблонов. В таблице 2 в качестве примера приведен шаблон КП для 4-х пользователей с указанием их логинов в системе. В шаблоне представлены статистические характеристики времени удержания (DT) в миллисекундах для нескольких букв русского алфавита.

Анализ полученных образцов КП показал, что временная характеристика DT для каждой клавиши имеет бимодальный закон распределения. На рисунке 7 гистограмма распределения приведена для клавиши А. Бимодальная форма объясняется наличием наложений при нажатии клавиш, то есть вторая клавиша уже нажата, а первая еще не отпущена. При этом время удержания увеличивается. Именно поэтому имеет смысл учитывать число наложений при распознавании КП.

Пример представления данных о КП пользователей, приведенный в табл. 1, демонстрирует определенное расхождение поведенческих характеристик пользователей. Визуально такое рас-

Таблица 3. Сравнительный анализ показателей эффективности

	Евклидово расстояние		Манхэттенское расстояние	
	без учета частотности	с учетом частотности	без учета частотности	с учетом частотности
FRR	2.8%	2.7%	2.6%	2.4%
FAR	0%	0%	0%	0%
FRR + FAR	13.6%	12.7%	13.3%	12.3%
Точность алгоритма	86.4%	87.3%	86.9%	87.7%

хождение между шестью пользователями видно на рис. 8 для всех букв русского алфавита, исключая редко используемые ф, ъ, ё.

Как видно из рис. 8 форма кривых, отражающая рисунок клавиатурного профиля разных пользователей, отличается друг от друга. Это подтверждает возможность распознавания ритма ввода пользователей, а именно ритм и определяет КП.

Также имеются незначительные расхождения между сеансами в клавиатурном профиле для отдельного пользователя. На рисунке 9а и 9б представлены DT для 10 сеансов двух пользователей домена.

Отчетливо видно, что в отдельных реализациях профиля есть визуальные отличия, но в целом КП каждого пользователя узнаваем, а, следовательно, отражает его поведенческую характеристику.

Отличия обусловлены тем, что клавиатурные нажатия производятся людьми, которые подвержены усталости, психоэмоциональным эмоциям или двигательным проблемам. Такие изменения указывают на то, что данные нажатия клавиш могут содержать шум и выбросы.

В ходе эксперимента текущие образцы КП, собранные клиентом программы, сопоставлялись с эталонными шаблонами БД на сервере. Для сравнения были вычислены приведенные метрические расстояния с учетом весовых коэффициентов частотности и при их отсутствии.

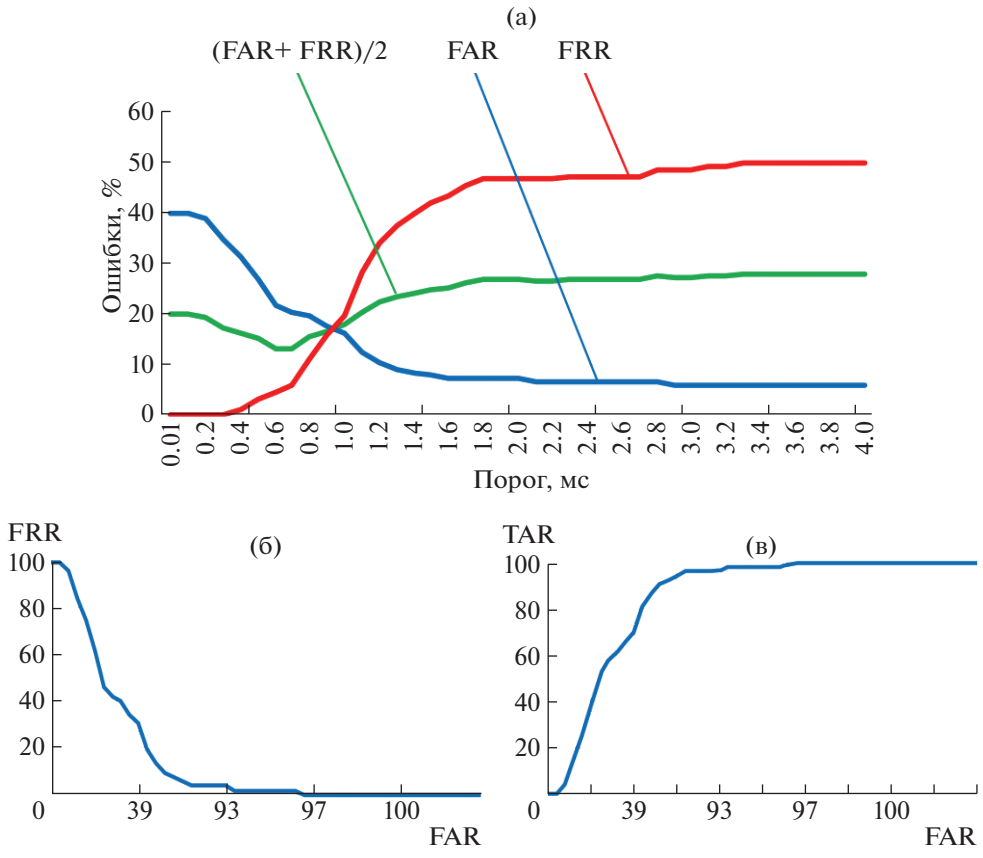


Рис. 10. Показатели эффективности распознавания пользователей.

Качество распознавания пользователей было оценено при помощи показателей эффективности, принятые в клавиатурной динамике и проанализированные в разделе 4. Часто используемые показатели FRR, FAR, и DET приведены соответственно на рисунке 10а, б, в. Также на рисунке 10а приведена полусумма показателей $\frac{FRR + FAR}{2}$, весьма привлекательная в ряде задач идентификации. Для установления значений порога сходства шаблонов в ходе эксперимента его значение варьировалось в диапазоне от 0.1 до 20 мс. На рисунке 10а приведена только информативная часть диапазона.

На основании проведенных исследований были получены значения минимальных критериев эффективности с использованием разных метрических расстояний и векторного критерия частотности букв алфавита, табл. 3.

В целом результаты получились довольно схожими для всех вариантов. Некоторое улучшение принесло использование масштабирующих коэффициентов учета частотности букв алфавита. В среднем суммарная ошибка уменьшилась на 1%, что отвечает основной цели исследования – повышение точности распознавания.

8. ЗАКЛЮЧЕНИЕ

Анализ исследований, проведенных нами и другими авторами в области идентификации пользователей на основе свободных текстов, позволил сделать ряд выводов.

1. Идентификация пользователя по его КП в режиме скрытого мониторинга возможна на основе произвольного текста, создаваемого в любом приложении.

2. Динамическая идентификация кроме особенностей, характерных для любого типа идентификации (создание шаблонов пользователей, выбор показателей, выбор алгоритмов распознавания) имеет ряд дополнительных:

– непрерывная корректировка динамических шаблонов, определяющая психоэмоциональное состояние человека;

– формирование векторного показателя клавиатурных нажатий, наиболее полно отражающего изменчивость клавиатурного почерка;

– включение в алгоритмы распознавания дополнительных классификационных признаков, например, связанных с частотностью использования букв в текстах или другими национальными особенностями текстов.

3. Исследования, проведенные в домене пользователей с хорошими навыками работы с компьютером, показали вполне удовлетворительную точность распознавания пользователей – в среднем 87%. Причем, точность не зависит от выбранного метрического расстояния при распознавании и несколько повышается при использовании масштабирующих коэффициентов учета частотности букв алфавита.

9. БЛАГОДАРНОСТИ

Работа выполнена при поддержке гранта РФФИ (№ 18-07-01007).

СПИСОК ЛИТЕРАТУРЫ

1. *Yampolskiy R.V.* Behavioural biometrics: a survey and classification / R. V. Yampolskiy, V. Govindaraju // International Journal of Biometrics. 2008. V. 1. № 1. P. 81–113.
2. *Jain A.* Handbook of biometrics / A. Jain, P. Flynn, A. Ross. N.Y.: Springer, 2007. 556 p.
3. *Васильев В.И., Ложников П.С., Сулавко А.Е., Еременко А.В.* Технологии скрытой биометрической идентификации пользователей компьютерных систем (обзор) // Вопросы защиты информации. 2015. № 3 (110). С. 37–47.
4. *Bergadano F., Gunetti D., Picardi, C.* User authentication through keystroke dynamics // ACM Transactions on Information and System Security. 2002. V. 5. № 4. P. 367–397.
5. *Karnan M., Akila M., Krishnaraj N.* Biometric personal authentication using keystroke dynamics: A review// Applied Soft Computing. 2011. V. 11. № 2. P. 1565–1573.
6. *Pisani P.H., Lorena A.C.* Emphasizing typing signature in keystroke dynamicsusing immune algorithms // Applied Soft Computing. 2015. V. 34. P. 178–193.
7. *Иванов А.И.* Биометрическая идентификация личности по динамике подсознательных движений. Пенза: ПГУ. 2000. 187 с.
8. *Chang T.Y.* Dynamically generate a long-lived private key based on password keystroke features and neural network // Information Sciences. 2011. V. 211. P. 36–47.
9. *Pisani P.H., Lorena A.C.* A systematic review on keystroke dynamics // Journal of the Brazilian Computer Society. 2013. V. 19. № 4. P. 573–587.
10. *Kim J., Kim H., Kang P.* Keystroke dynamics-based user authentication using freely typed text based on user-adaptive feature extraction and novelty detection // Applied Soft Computing. 2018. V. 62. P. 1077–1087.
11. *Gunetti D., Picardi C.* Keystroke analysis of free text // ACM Transactions on Information and System Security. 2005. V. 8. P. 312–347.
12. *Messerman T., Mustafić S., Camtepe A., Albayrak S.* Continuous and non-intrusive identity verification in real-time environments based on free-text keystroke dynamics // Proceedings of the International Joint Conference on Biometrics (IJCB 2011). 2011. P. 1–8.
13. *Alsultan A., Warwick K., Wei H.* Non-conventional keystroke dynamics for user authentication // Pattern Recognition Letters. 2017. V. 89. P. 53–59.
14. *Kang P., Cho S.* Keystroke dynamics-based user authentication using long and free text strings from various input devices // Information Sciences. 2015. V. 308. P. 72–93.

15. *Ahmed A.A.* Biometric recognition based on free-text keystroke dynamics // IEEE Transactions on Cybernetics. 2014. V. 44. № 4. P. 458–472.
16. *Alsultan K., Warwick H.* Keystroke dynamics authentication: a survey of free-text methods // International Journal of Computer Science Issues. 2013. V. 10. № 4. P. 1–10.
17. *Joyce R., Gupta G.* Identity authentication based on keystroke latencies // Communications of the ACM. 1990. V. 33. № 2. P. 168–176.
18. *Spillane R.J.* Keyboard Apparatus for Personal Identification // Technical Disclosure Bulletin. 1975. V. 17. № 3346.
19. *Crawford H.* Keystroke dynamics: Characteristics and opportunities // Proceedings of the Eighth Annual International Conference on Privacy Security and Trust (PST). 2010. P. 205–212.
20. *Peacock A., Ke X., Wilkerson M.* Typing patterns: A key to user identification // IEEE Security and Privacy. 2004. V. 2. № 5. P. 40–47.
21. *Shanmugapriya D., Padmavathi G.* A survey of biometric keystroke dynamics: Approaches, security and challenges // International Journal of Computer Science and Information Security. 2009. V. 5. № 1. P. 115–119.
22. *Banerjee S.P., Woodard D.L.* Biometric authentication and identification using keystroke dynamics: a survey // Journal of Pattern Recognition Research. 2012. V. 7. № 1. P. 116–139.
23. *Teh P.S., Teoh A.B., Yue S.* A survey of keystroke dynamics biometrics // The Scientific World Journal. 2013. P. 1–24.
24. *Mondal S., Bours P.* A study on continuous authentication using a combination of keystroke and mouse biometrics // Neurocomputing. 2016. V. 230. P. 1–22.
25. *Крутохвостов Д.С., Хиценко В.Е.* Парольная и непрерывная аутентификация по клавиатурному почерку средствами математической статистики // Вопросы кибербезопасности. 2017. Т. 24. № 5. С. 91–99.
26. *Kochegurova E.A., Luneva E.E., Gorokhova E.S.* On continuous user authentication via hidden free-text based monitoring // Advances in Intelligent Systems and Computing. 2019. V. 875. P. 66–75.
27. *Vinayak R., Arora K.* A Survey of User Authentication using Keystroke Dynamics // International Journal of Scientific Research Engineering & Technology (IJSRET). 2015. V. 4. № 4. P. 378–384.
28. *Teh P.S.* A survey on touch dynamics authentication in mobile devices. Review article / P.S. Teh, N. Zhang, A.B. Teoh, K. Chen // Computers & Security. 2016. V. 59. P. 210–235.
29. *Mahfouz A., Eldin A.S., Mahmoud T.M.* A survey on behavioral biometric authentication on smartphones // Research article Journal of Information Security and Applications. 2017. V. 37. P. 28–37.
30. *Corpus K.R., Gonzales R.J., Morada A.S., Vea L.A.* Mobile user identification through authentication using keystroke dynamics and accelerometer biometrics // Proceedings of the International Conference on Mobile Software Engineering and Systems (MOBILESoft 16). 2016. P. 1–12.
31. *Соколов Д.А.* Использование клавиатурного почерка для аутентификации в распределенных системах с мобильными клиентами // Безопасность информационных технологий. 2010. № 2. С. 50–53.
32. *West A.G.* Analyzing the Keystroke Dynamics of Web Identifiers // Proceedings of the 2017 ACM on Web Science Conference (WebSci'17). 2017. P. 181–190.
33. *Pentel A.* Predicting Age and Gender by Keystroke Dynamics and Mouse Patterns // Proceedings of UMAP'17 Adjunct, Publication of the 25th Conference on User Modeling. Adaptation and Personalization. 2017. P. 381–385.
34. *Ложников П.С., Сулавко А.Е., Бурая Е.В., Писаренко В.Ю.* Аутентификация пользователей компьютера на основе клавиатурного почерка и особенностей лица // Вопросы кибербезопасности. 2017. Т. 21. № 3. С. 24–34.
35. *Morales A., Fierrez J., Tolosana R., Ortega-Garcia J., Galbally J., Gomez-Barrero M., Anjos A., Marcel S.* KBOC: Keystroke Biometrics OnGoing Competition // Proceedings 8th IEEE International Conference on Biometrics: Theory, Applications and Systems. 2016. P. 1–6.
36. *Ворона В.А., Тихонов В.А.* Системы контроля и управления доступом. М.: Горячая линия – Телеком, 2010, 274 с.
37. *Berthold M., Borgelt C., Hopner F., Klawonn F.* Guide to Intelligent Data Analysis. Texts in Computer Science, London: Springer, 2010. V. 42. 394 p.
38. *Gaines R.S., Lisowski W., Press S.J., Shapiro N.* Authentication by Keystroke Timing: Some Preliminary Results. Technical Report R-2526-NSF. Santa Monica. CA: Rand Corporation, 1980. 41 p.
39. *Ali M.L., Monaco J.V., Tappert C.C., Qiu M.* Keystroke Biometric Systems for User Authentication // J Sign Process Systems. 2017. V. 86. P. 175–190.
40. *Kochegurova E.A., Gorokhova E.S., Mozgaleva A.I.* Development of the Keystroke Dynamics Recognition System // Journal of Physics: Conference Series. 2017. V. 803. № 1. P. 1–6.
41. *Alpar O.* Frequency spectrograms for biometric keystroke authentication using neural network based classifier // Knowledge-Based Systems. 2017. V. 116. P. 163–171.
42. *Goodkind A., Brizan D.G., Rosenberg A.* Utilizing overt and latent linguistic structure to improve keystroke-based authentication // Image and Vision Computin, 2017. V. 58. P. 230–238.
43. *Dozono H., Ito S., Nakakuni M.* The authentication system for multi-modal behavior biometrics using concurrent Pareto learning SOM // Proceedings of the 21st International Conference on Artificial Neural Networks. 2011. Part II. P. 197–204.
44. *Popovici E.C., Guta O.G., Stancu L., Arseni S.C., Fratu O.* Mlp neural network for keystroke-based user identification system // Proceeding 11th international conference on telecommunication in modern satellite, cable and broadcasting services (TELSIKS). 2013. V. 1. P. 155–158.
45. *Maxion R.A., Killourhy K.S.* Keystroke biometrics with number-pad input // Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN '10). 2010. P. 201–210.

ПРОГРАММИРОВАНИЕ, 2020, № 1, с. 29–38

ПРОГРАММНАЯ ИНЖЕНЕРИЯ, ТЕСТИРОВАНИЕ И ВЕРИФИКАЦИЯ ПРОГРАММ

УДК 004.42

АЛГОРИТМ КЛАСТЕРИЗАЦИИ МНОЖЕСТВА ДЕТАЛЕЙ ПО ЧЕРТЕЖАМ

© 2020 г. В. Н. Кучуганов[a],*, А. В. Кучуганов[a],, Д. Р. Касимов[a],*****

[a] *ФГБОУ ВО "Ижевский государственный технический университет им. М.Т. Калашникова"*
426069 Ижевск, ул. Студенческая, 7, Россия

**E-mail: kuchuganov@istu.ru*

***E-mail: Aleks_KAV@udm.ru*

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

Поступила в редакцию 15.01.2019 г.
После доработки 19.05.2019 г.
Принята к публикации 27.05.2019 г.

В статье для задач производственной логистики предлагается алгоритм кластеризации заданного множества деталей по чертежам. С целью ускорения процесса выполняется фаззификация значений каждого параметра и поиск локальных максимумов на N-мерной гистограмме, отличающийся способом выборки соседних векторов, который состоит в пересчете координат из N-мерного пространства в одномерное и обратно и сравнении с координатами соседних векторов. За счет этого алгоритм осуществляет поиск вершин кластеров за один проход и не требует создания множества локальных списков или графов соседства векторов, благодаря чему повышается эффективность алгоритма кластеризации. Приведены результаты эксперимента по автоматическому группированию деталей на основе чертежей.

DOI: 10.31857/S0132347420010045

1. ВВЕДЕНИЕ

Классификаторы деталей достаточно широко применяются в машиностроении с целью унификации конструкций, технологий и технологической оснастки [1]. Но применение их для разработки групповых технологий, обеспечивающих повышение эффективности производства, сдерживается проблемами автоматизации процесса группирования (кластеризации) деталей по их чертежам или 3D геометрическим моделям. Оперативная кластеризация деталей по геометрическим, техническим, технологическим, производственным параметрам могла бы повысить степень унификации технологических процессов, повысить обоснованность и качество нормирования труда, оптимизировать планы производства. Для этого необходимо привлечение методов Image Mining – автоматического извлечения признаков формы и относительного расположения структурных элементов из чертежей, 3D геометрических моделей, технологических эскизов.

На современных предприятиях применяются системы класса PLM (Product Lifecycle Management), решающие задачи накопления и управления всей информацией, возникающей на этапах конструкторского и технологического проектирования, изготовления, сопровождения и т.д. [2]. Благодаря этому практически у каждого чертежа имеется подробный "паспорт", что позволяет использовать эту информацию и для автоматизации группирования деталей.

Целью данной работы является создание эффективного алгоритма кластеризации машиностроительных деталей по набору параметров, включающего:

– геометрические признаки, которые могут быть извлечены из чертежей деталей с помощью алгоритмов автоматической векторизации чертежей, структурного анализа и распознавания типовых конструктивных элементов (ТКЭ);

– дополнительные технологические параметры, которые могут быть получены из электронного архива чертежей.

Особенностью предлагаемого алгоритма является фаззификация значений каждого параметра, что помимо ускорения процесса кластеризации также дает пользователю отчетливую обратную связь в качественных терминах.

Статья организована следующим образом. В разделе 2 анализируются существующие методы кластеризации объектов, в том числе по параметрам формы. В разделе 3 описывается укрупненный алгоритм кластеризации. В разделе 4 рассматриваются функции пересчета координат из одномерного массива в многомерное пространство и обратно, посредством которых устанавливается соседство (близость) объектов. В разделе 5 описа-

ны эксперименты по автоматическому группированию деталей на основе чертежей. Заключение по работе дается в разделе 6.

2. ОБЗОР МЕТОДОВ КЛАСТЕРИЗАЦИИ

В настоящее время существует множество методов кластерного анализа. С их помощью аналитики получают возможность классифицировать информацию об объектах, т.е. на основе известной информации группировать объекты в целях анализа, лучшего понимания, принятия управленческих решений.

Наиболее популярными можно считать методы, использующие подход на основе k-средних, и гистограммные методы.

Алгоритм кластеризации методом k-средних (k-means) [3] минимизирует среднеквадратичное отклонение векторов признаков с целью получения k-кластеров – компактных множеств объектов, близких по свойствам. Алгоритм является итерационным, т.е. требует значительных ресурсов. Недостатком считается отсутствие возможности определить реально существующее количество кластеров. Чтобы уменьшить объем вычислений обычно сокращают количество итераций, но при этом появляется риск пропустить кластеры, важные для решаемой прикладной задачи.

Гистограммные методы отыскивают все вершины кластеров за один проход, но это требует слишком больших вычислений. Чтобы ускорить процесс, предварительно осуществляют квантование пространства путем огрубления значений параметров. Так, в работе [4] предлагается осуществлять разбиение векторного пространства на множество подпространств и строить многомерную гистограмму полученных векторов, исключая пустые подпространства. Список этих векторов (подпространств) организован по типу алфавитного словаря, т.е. в порядке возрастания значений цифр очередной координаты. Для удобства анализа соседних подпространств строится множество локальных графов векторов и их соседей. Далее применяется гистограммный алгоритм Нарендры [5], не требующий задания числа кластеров или количества итераций. Время классификации линейно зависит от числа различных векторов. С помощью алгоритма осуществлялась сегментация областей на спутниковых снимках по ряду спектральных каналов.

Кластеризация широко применяется для сегментации изображений по цветовым характеристикам пикселей. Так, в статье [6] проблема сегментации изображения решается посредством иерархической последовательности кусочно-постоянных приближений с использованием метода кластеризации Уорда [7], применяющего принципы дисперсионного анализа для оценки расстояний между кластерами. Мерой различия объектов служит евклидово расстояние, которое, к сожалению, невозможно применить к атрибутам, которые по своей сути являются качественными и не могут быть выражены в количественной шкале. А в задаче группирования деталей такого рода атрибуты (например: вид детали, тип заготовки, группа материала и т.п.) имеют существенное значение, и их не следует игнорировать.

Нечеткие классификаторы часто применяют в системах поддержки принятия управленческих решений при анализе больших объемов данных. В работе [8] задачу построения нечетких классификаторов, формулируемую как задачу отбора признаков, генерации базы нечетких правил, оптимизации условий правил, предлагается решать путем отбора классифицирующих признаков на основе бинарного гармонического поиска [9], использования метаэвристических алгоритмов непрерывного гармонического поиска, являющихся дальнейшим развитием эволюционных вычислений, а также путем генерации базы правил по экстремальным значениям признаков. Эффективность предлагаемого подхода проверена на реальных данных из KEEL и KDD Cup хранилищ текстовых сообщений. К сожалению, для построения классификаторов в задачах производственной логистики такой принцип, как правило, не подходит, поскольку производственники – конструкторы, технологи, плановики – обычно сами решают (назначают) критерии группирования, которые они считают важнейшими в той или иной производственной ситуации.

Для кластеризации объектов по параметрам формы обычно используют статистические характеристики, в частности, скелетные и контурные дескрипторы. Так в работе [10] для повышения надежности кластеризации предлагается алгоритм спектральной кластеризации, основанный на спектральном анализе множества пикселей, составляющих изображение. Алгоритм не требует представления формы в виде вектора.

В работе [11] предлагается осуществлять кластеризацию плоских фигур путем формирования графов скелетонов и обнаружения общей структуры фигур на основе результатов сопоставления графов. Используется стандартная агломеративная итерационная схема кластеризации. В каждой итерации графы скелетонов из двух кластеров объединяются в один граф, вычисляется мера сходства в качестве критерия близости кластеров. Эксперименты выполнены на популярных тестах для форм – рисунках людей, животных, рыб и т.п. – проблема весьма актуальная в задачах поиска изображений по контенту (CBIR – Content-Based Image Retrieval).

Коммерчески предлагаемая система CADFind [12, 13] способна выполнять поиск машинострои-

тельных чертежей и 3D моделей по произвольному образцу, задаваемому пользователем, а также производить автоматическое группирование деталей по их чертежам и 3D моделям. Система автоматически идентифицирует потенциальные семейства деталей и позволяет уточнять их в интерактивном режиме. В доступных описаниях принципов работы этой системы указывается, что она индексирует геометрию детали, а также текстовую информацию, связанную с ней, включая материал, техпроцесс. Извлекаемая из чертежа/3D модели информация представляется в виде кода групповой технологии (GT кода) – строки буквенно-цифровых символов. Сопоставление деталей происходит путем нечеткого сравнения соответствующих GT кодов. К сожалению, ни на официальном сайте продукта, ни в публикациях авторов нам не удалось найти более подробной информации о том, какие именно геометрические признаки детали описывает формируемый GT код, каким образом осуществляется автоматическое извлечение этих признаков из чертежей и 3D моделей, в чем заключается нечеткий характер процедуры сравнения GT кодов. Не раскрывается также и применяемый метод кластеризации, на котором базируется процесс автоматического группирования деталей. Остается только сделать вывод о том, что проблема весьма актуальна, а нацеленные на ее решение программные системы – востребованы.

В части контентного поиска машиностроительных чертежей по образцу существуют также другие исследовательские работы (например, [14, 15]). Однако в этих работах не делается акцент на задаче автоматической классификации деталей.

3. УКРУПНЕННЫЙ АЛГОРИТМ КЛАСТЕРИЗАЦИИ

Предлагаемый алгоритм кластеризации относится к категории гистограммных и состоит в следующем:

1. Подготовка исходных данных. Подбор репрезентативной выборки объектов, заполнение массива измерений (таблица 1), где M – количество объектов в обучающей выборке, N – количество параметров.

2. Фаззификация значений атрибутов, описывающих объекты. Предварительная (перед собственно кластеризацией) фаззификация параметров существенно сокращает объем вычислений.

В результате фаззификации для каждого параметра p_i, $i \in \{1 \ldots N\}$ с диапазоном значений D_i получаем линейно упорядоченное множество качественных значений параметра $F_i = \{p_{ij}\}$, где $j \in \{1 \ldots h(i)\}$, $h(i)$ – количество качественных значений параметра p_i и (сюръективную) функцию val_i: $D_i \to F_i$, которая сопоставляет каждому возможному значению параметра p_i соответствующее качественное значение. За счет этого диапазон значений каждого параметра p_i разбивается на конечное количество качественных значений $D_{ij} = \{x \in D_i | val_i(x) = p_{ij}\}$.

Таблица 1. Структура массива измерений

Объекты	Параметры			
	1	2	†	N
1				
2				
.				
.				
.				
M				

3. Построение N-мерной гистограммы. Гиперпространство содержит не более чем H^N локусов (ячеек, подпространств), где H – максимальное количество различных качественных значений параметра (например, по каждому из параметров может быть получено одно из $H = 5$ возможных качественных значений). В цикле по массиву измерений сформировать массив локусов (таблица 2), где для каждого локуса указать количество объектов в нем, т.е. объектов, имеющих соответствующие качественные значения всех N параметров. В массиве измерений отметить, к какому локусу относится объект. Номер кластера на данном этапе отсутствует.

4. Поиск локальных максимумов в N-мерном пространстве измерений. Для этого с помощью ***локального анализатора окрестности локуса*** просканировать пространство и найти такие локусы, в которых по каждому из N направлений выполняется условие:

$$\forall l = 1..L(\exists i \in \{i_1, \ldots, i_N\}(w(F1(i_1,\ldots,h(i)-1,\ldots,i_N)) < w(l) \geq w(F1(i_1,\ldots,h(i)+1,\ldots,i_N)))) \Rightarrow L_l \in \{C\}, \quad (1)$$

где: $l = 1..L$ – индекс локуса в одномерном массиве L локусов;

$F1(i_1,...,i_N)$ – функция, вычисляющая положение локуса в одномерном массиве L, где $i_1, \ldots, i_N$ – координаты локуса в N-мерном пространстве измерений;

$h(i) = 0..H$ – положение локуса вдоль i-той оси координат, $i \in \{i_1, \ldots, i_N\}$;

$w(l)$, $w(F1)$ – вес локуса (количество объектов, по своим значениям попавших в соответствующее подпространство);

$\{C\}$ – множество локальных максимумов.

Найденные локусы в гиперпространстве параметров служат "вершинами" кластеров – скоплений объектов с близкими характеристиками. Получаем список кластеров

Таблица 2. Массив локусов

Локус	Координаты (качественные) локуса				№ кластера	Вес (количество объектов)
	1	2	...	N		
1					1	w(1)
.					.	.
.					.	.
.					.	.
L					K	w(L)

$$c_k = \langle w(k), x_1, \ldots, x_N \rangle, \quad k = 1..K,$$

где: x_1, ..., x_N – координаты вершины кластера, удовлетворяющего условию (1) в N-мерном пространстве.

5. Определение границ кластеров. Для этого из вершины кластера c_k, имеющего координаты x_1, ..., x_N, выше описанный локальный анализатор запускается рекурсивно. Если вес $w(F1(x_1, ..., h(i) \pm 1, ... x_N))$ соседнего по какому-либо i-тому направлению локуса меньше или равен весу локуса, принадлежащего кластеру, то этот соседний локус присоединяется к тому же кластеру:

$$\forall l \in \{1..H^N\}(\exists i \in \{x_1, \ldots, x_N\}$$
$$w(F1(x_1, \ldots, h(i) \pm 1, \ldots x_N))$$
$$\leq w(F1(x_1, \ldots, x_N)))$$
$$\Rightarrow L(F1(x_1, \ldots, h(i) \pm 1, \ldots x_N)) \in c_k;$$

где: N – количество параметров;

$h(i) = 0..H$ – положение локуса вдоль i-той оси координат, $i \in \{x_1, ..., x_N\}$;

H – количество возможных значений i-го параметра;

l – индекс локуса в одномерном массиве локусов L;

$L(F1(x_1, ..., h(i) \pm 1, ... x_N))$ – локус соседний с анализируемым;

c_k – кластер с номером k.

Очевидно, таким же образом процесс кластеризации можно осуществлять в каждом из полученных кластеров – фаззифицировать и кластеризовать, по вышеописаному алгоритму не всю "обучающую" выборку, а уже полученный кластер, что дает возможность более детально анализировать интересующие области пространства измерений и детализировать классификацию объектов.

Функция, определяющая принадлежность нового объекта тому или иному кластеру – это мера относительной близости, вычисляемая как расстояние в N-мерном метрическом пространстве между объектом и каждым из "центров" кластеров.

4. ФУНКЦИИ ПЕРЕСЧЕТА КООРДИНАТ

Пусть N – количество осей пространства параметров, H – количество возможных значений каждого параметра.

Создадим одномерный (линейный) массив L локусов. Количество элементов в массиве будет H^N.

Координаты $\langle x_1, x_2, ..., x_N \rangle$ локуса (ячейки) в N-мерном пространстве запишем в виде числа в позиционной системе счисления с основанием H. Это число содержит N цифр, а каждая цифра может принимать значения от 0 до $H - 1$.

Чтобы вычислить значение индекса l элемента в одномерном массиве L, достаточно отобразить координаты локуса в H-ричное число и перевести это число из позиционной системы счисления с основанием H в десятичную систему счисления. И наоборот, если известен номер локуса в одномерном массиве, то для определения его координат в N-мерном пространстве нужно перевести этот номер в H-ричное число. Для выборки ближайших к нему соседних по i-той оси локусов нужно увеличить или уменьшить на 1 i-тую цифру H-ричного числа и полученные числа перевести в десятичные. Чтобы проверить всех ближайших соседей, то же самое надо сделать по всем другим цифрам H-ричного числа.

Если в одномерном массиве не хранить пустые локусы, то в оставшихся должны быть в качестве ключа записаны их координаты с целью извлечения нужных локусов. Выбор того или другого способа хранения информации зависит от ожидаемого числа пустых ячеек.

С другой стороны, существование пустых локусов может оказаться интересным для какой-то предметной области, например, в социологических исследованиях, где параметры, как правило, качественные.

Перевод H-ричного числа в десятичную систему и обратно (см. например, [16]) осуществляется по формулам:

$$X = \sum_{k=-n}^{m} R_k * B^k, \qquad (2)$$

где X – значение числа в десятичной системе, B – основание H-ричной системы счисления, R – цифра H-ричного числа, k – позиция цифры в записи числа.

Алгоритм поиска центров кластеров выглядит следующим образом:

```
k := 0;
FOR l = 1..L // Цикл по всем локусам, L = H^N
    IF p[l] = 0 THEN Continue(l); // Переход к следующему l
    FN(l, ZH); // Перевод l → Z_H в H-ричную систему счисления (с основанием H)
    // Сформировать массив (или список) из цифр числа ZH
    FOR i = 1..N // Цикл по цифрам числа ZH
        // Сформировать число ZL такое, что i-тая цифра ZL меньше на 1,
        // чем i-тая цифра в ZH
        F1(ZL, ll); // Выдает индекс ll соседнего локуса в одномерном массиве L
        IF p[l] < p[ll] THEN Continue(l); // Переход к следующему l
        // Сформировать ZR такое, что i-тая цифра ZR больше на 1, чем i-тая цифра в ZH
        F1(ZR, lr); // Выдает индекс lr соседнего локуса в одномерном массиве L
        IF p[l] ≤ p[lr] THEN Continue(l); // Переход к следующему l
    END i
    k := k + 1; c[k] := ⟨ p[l], l, ZH⟩ // Центр k-го кластера
END l
```

Функции FN, $F1$ – это несколько команд. Можно не искать границы кластеров. Обычно объект относят к тому центру, до которого ближе.

Сложность алгоритма не превосходит значения функции O:

$$O(H^N \times N),$$

где N – количество параметров;

H – максимальное количество различных качественных значений параметра.

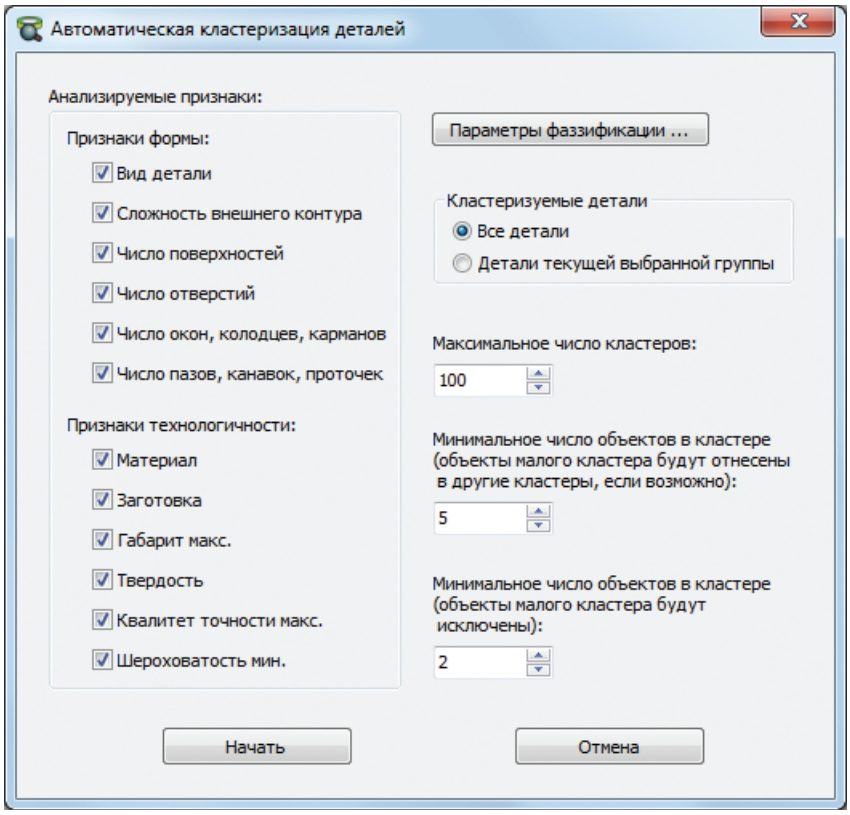
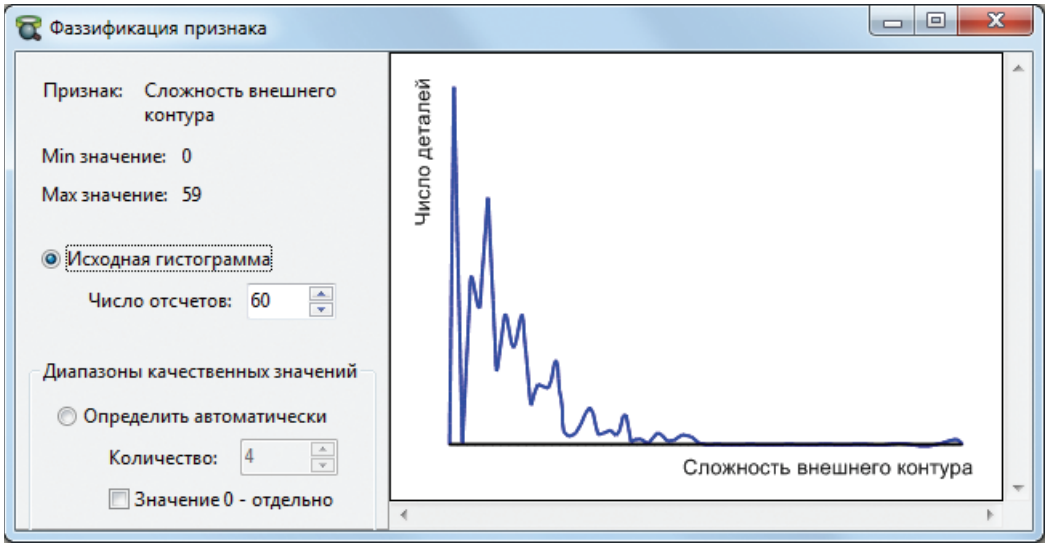


Рис. 1. Окно настроек кластеризации.

(а)



(б)

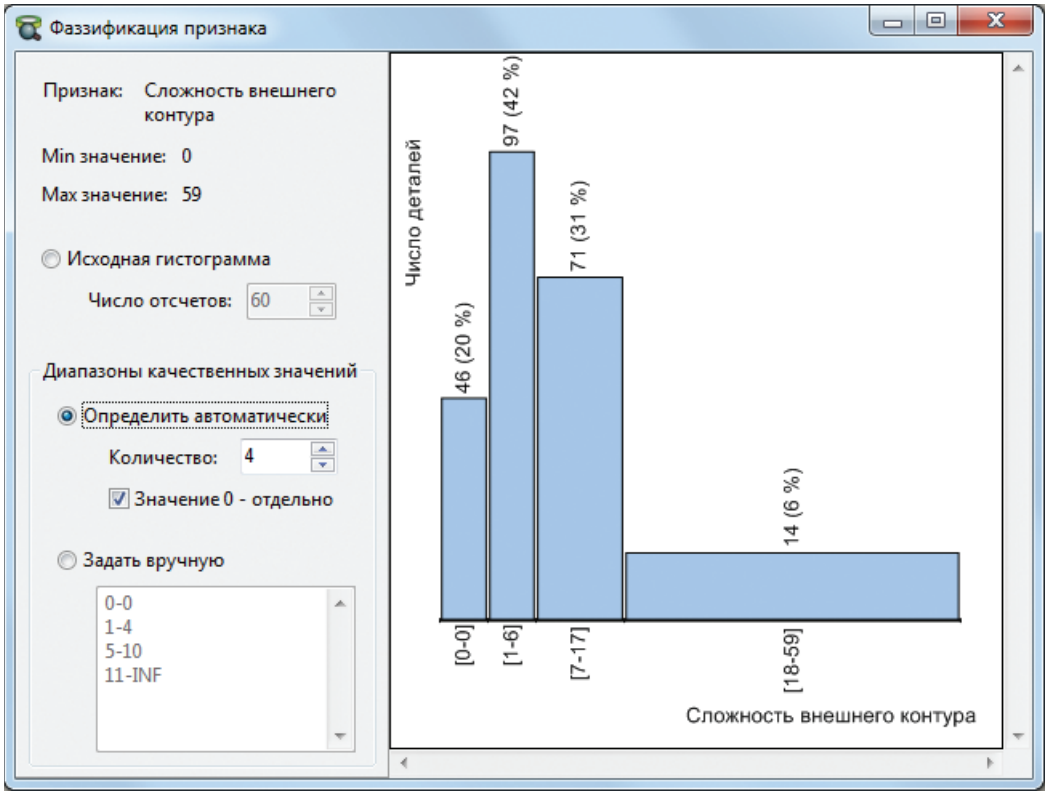


Рис. 2. Инструментарий перевода количественных значений признака в качественные: а) исходная гистограмма распределения количественных величин; б) диапазоны качественных значений, предложенные системой (автоматически).

В практических задачах это зачастую оказывается меньше, чем линейная зависимость от числа анализируемых объектов, поскольку редко требуется оперировать более чем 5 качественными значениями параметра, и числом анализируемых параметров обычно управляет лицо, принимающее решение, с целью повышения наглядности результатов. А если параметров много, то применяют *факторный анализ*, заменяя группу свойств одним обобщенным свойством.

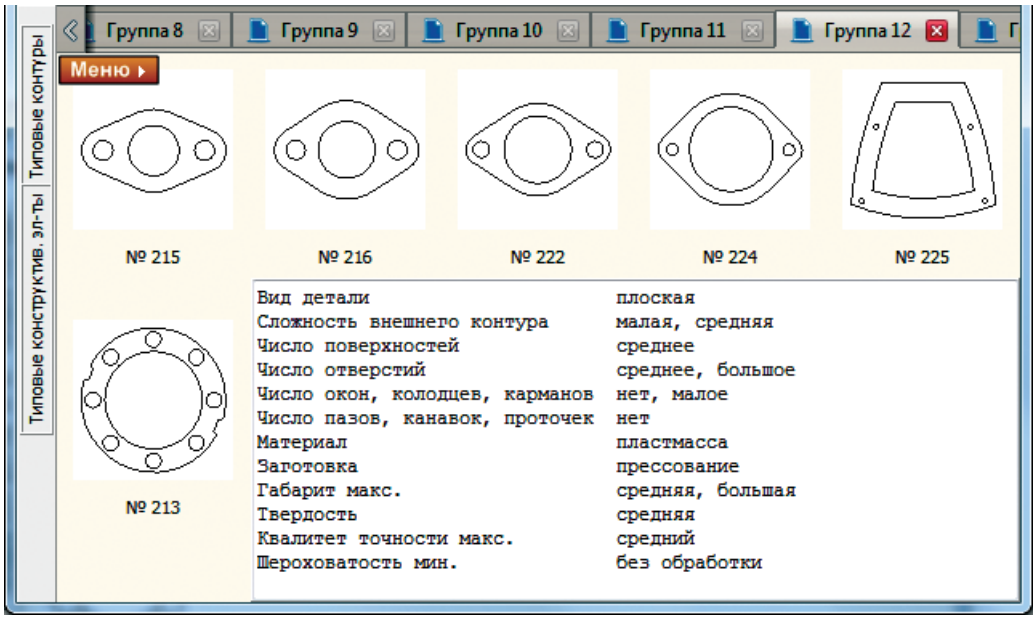


Рис. 3. Вывод результатов кластеризации.

5. ЭКСПЕРИМЕНТ

Алгоритм кластеризации, описанный выше, был использован для синтеза классификатора деталей.

В эксперименте анализировались такие параметры деталей как: вид детали (корпусная, плоская, тело вращения); сложность внешнего контура (количество перегибов – смен знака приращения угла наклона); число поверхностей, рассчитываемое на основе выделения минимальных примыкающих контуров; количество простых и ступенчатых отверстий; количество окон, колодцев и карманов; количество пазов, канавок и проточек. Дополнительные параметры классификации: группа материала (сталь, чугун, силумин, пластмасса); заготовка (пруток, плита, литье, прессование, прокат); габариты; твердость; квалитет точности; шероховатость (без обработки, шлифование/точение, полирование, зеркальная). Общее количество анализируемых параметров – 12. Числовые параметры преобразовывались в качественные значения (нет, мало, средне, много).

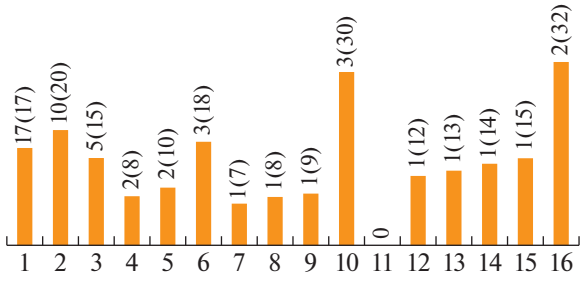


Рис. 4. Распределение деталей по кластерам (по горизонтальной оси – размер кластера, по вертикальной оси – число кластеров и общее количество деталей в них).

На рис. 1 показано окно запуска процесса кластеризации деталей. Здесь пользователь, обычно, конструктор-технолог, выбирает анализируемые признаки, с помощью инструмента “Параметры фаззификации” корректирует функции преобразования количественных значений признаков в качественные, указывает множество кластеризуемых деталей и задает ограничения на количество и размер целевых кластеров. Чем больше кластеров он хочет получить, тем точнее будет выполнена кластеризация. Соответственно, придется разрабатывать больше групповых техпроцессов.

Процесс перевода количественных значений признаков в качественные является одним из инструментов управления точностью и производительностью процесса кластеризации. На рис. 2 показано окно программы, содержащее средства анализа отдельного признака. Первое, что видит эксперт в данном окне – гистограмму распределения значения признака в выборке деталей (рис. 2а). По нажатию на какой-либо отсчет гистограммы, на экране отображаются примеры деталей, имеющих соответствующее значение признака. Получив общую характеристику набора данных, эксперт затем определяет:

1) число качественных значений признака, которыми целесообразно оперировать в процессе кластеризации;

2) соответствия между качественными и количественными значениями.

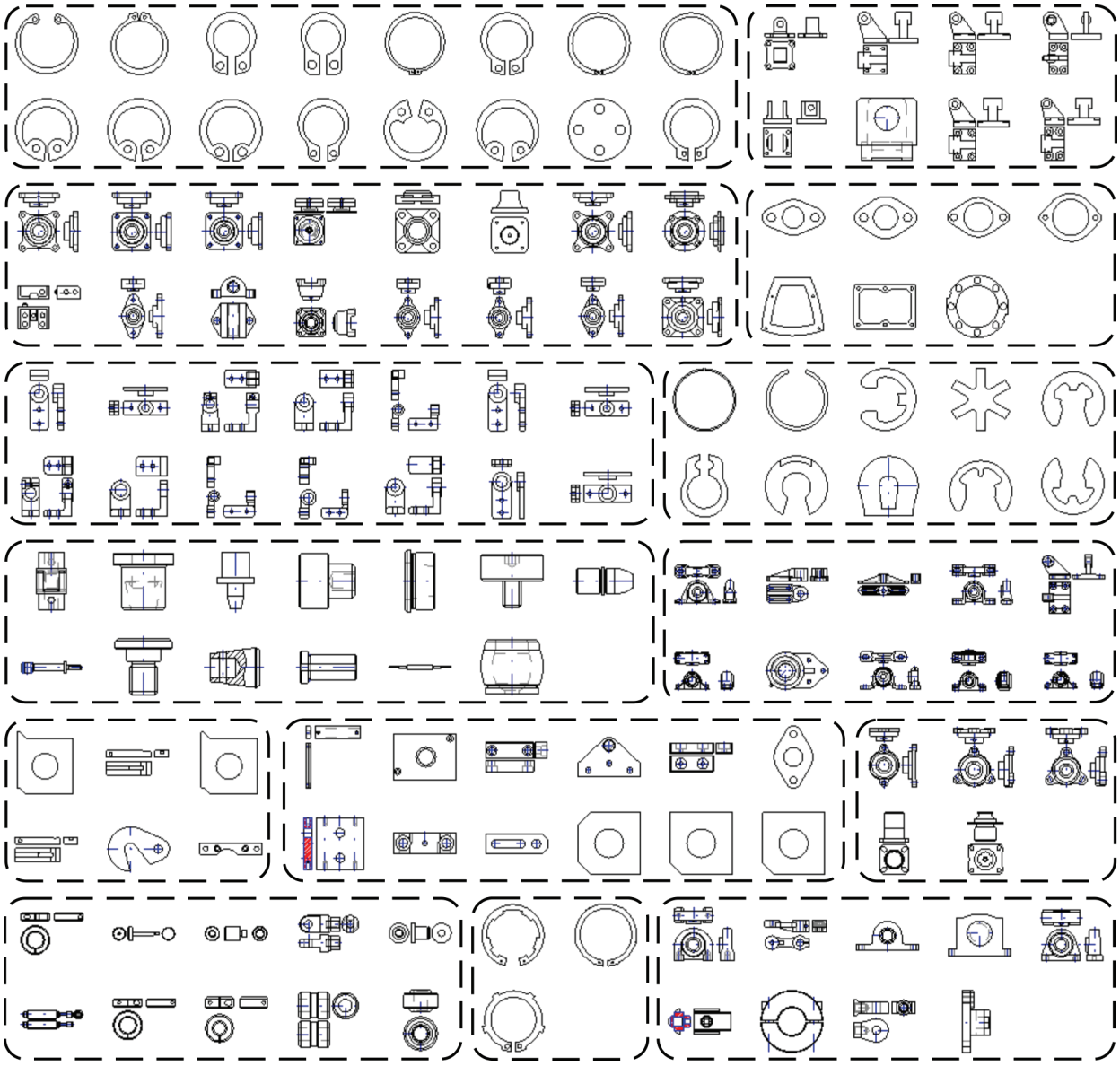


Рис. 5. Примеры полученных кластеров.

Качественное значение представляет определенный диапазон количественных величин. Основная задача, возлагаемая на эксперта, состоит в установлении границ диапазонов. Для упрощения этого процесса, мы предусмотрели возможность автоматического получения рекомендуемых границ диапазонов (рис. 2б). Программа выявляет на исходной гистограмме локальные максимумы (в количестве, равном заданному числу целевых диапазонов). В примере на рис. 2б были получены следующие локальные максимумы (отсчеты): 0, 4, 12, 25. Затем к ним программа относит остальные отсчеты гистограммы по критерию близости. В результате формируется опорная разбивка на диапазоны, которую эксперт может затем подкорректировать в ручном режиме.

На рис. 3 приведен пример результата кластеризации. При визуализации полученных кластеров помимо чертежей деталей отображаются также значения их признаков в качественном представлении. Это помогает инженеру интерпретировать результаты.

Для экспериментального исследования был подготовлен набор данных, содержащий 98 чертежей корпусных деталей, 82 чертежа плоских деталей и 48 чертежей тел вращения, всего 228 деталей. Чертежи деталей и часть данных о материале и габаритах взяты из открытых онлайн-каталогов (TraceParts, PARTcommunity, Mitutoyo) и архивов

студенческих работ. Значения дополнительных технологических параметров, отсутствующие в первоисточнике, задавались вручную экспертом-технологом. При необходимости выполнялась векторизация сканированных чертежей [17]. Извлечение признаков формы и распознавание ТКЭ на чертежах осуществляла разработанная подсистема графического поиска чертежей [18].

Синтаксическое распознавание ТКЭ на чертежах осуществлялось с помощью порождающих грамматик, их описывающих. Ниже показаны примеры функций описания конструктивных элементов деталей: “окно” – сквозная прорезь в стенке, “колодец”, “карман” – углубление, у которого один край совпадает с краем детали:

<*Элемент*> ::= $type, x_s, y_s, x_e, y_e, x_c, y_c, r, \alpha, \beta$,
где $type$ – тип элемента, $type \in \{Line, Arc, Circle\}$;
x_s, y_s – координаты точки начала элемента;
x_e, y_e – координаты точки конца элемента;
x_c, y_c – координаты центра дуги/окружности;
r – радиус дуги/окружности;
α, β – углы векторов от центра к концам дуги.

<*Цепочка элементов*> ::= $e_1, e_2, ..., e_k$: <*Усл*1>, <*Усл*2>,
где <*Усл*1> ::= $\forall i \in [1..k]$ (e_i *is* <*Элемент*> $\wedge$ e_i *.type* $\neq$ *Circle*);
<*Усл*2> ::= $\forall i \in [1..k-1]$ (*Соединяются*(e_i, e_{i+1}) $\wedge$ $\forall j \in [i+2..k] \neg$ *Пересекаются*(e_i, e_j)).

<*Контур*> ::= (<*Цепочка элементов*> : <*Усл*1>) | (<*Элемент*> : <*Усл*2>),
где <*Усл*1> ::= *Соединяются*(e_k, e_1),
<*Усл*2> ::= *type* = *Circle*.

<*Узел*> ::= e_1, e_2, e_3 : <*Усл*1>,<*Усл*2>,
где <*Усл*1> ::= $\forall i \in [1..3]$ e_i *is* <*Элемент*>;
<*Усл*2> ::= *Соединяются*(e_1, e_2) $\wedge$ *Соединяются*(e_2, e_3) $\wedge$ *Соединяются*(e_1, e_3).

<2*СвязанныхКонтура*> ::= C_1, C_2: <*Усл*1>, <*Усл*2>,
где <*Усл*1> ::= (C_1 *is* <*Контур*>) $\wedge$ (C_2 *is* <*Контур*>);
<*Усл*2> ::= *ПроекционноСвязаны*(C_1, C_2).

<*Окно*> ::= <2*СвязанныхКонтура*>: <*Усл*1>, ..., <*Усл3*>,
где <*Усл*1> ::= $\neg\exists U$ (U *is* <*Узел*> $\wedge$ $U \cap C_1 \neq \varnothing$);
<*Усл*2> ::= $\forall i \in [1..k]$ $e_i \in C_1$ (*ПриращениеУглаВектораКасательной*(e_i, e_{i+1}) ≤ 0);
<*Усл3*> ::= $\exists_{\ge 4} U$ (U *is* <*Узел*> $\wedge$ $U \cap C_2 \neq \varnothing$).

<*Колодец*> ::= <2*СвязанныхКонтура*>: <*Усл*1>, ..., <*Усл3*>,
где <*Усл*1> ::= $\neg\exists U$ (U *is* <*Узел*> $\wedge$ $U \cap C_1 \neq \varnothing$);
<*Усл*2> ::= $\forall i \in [1..k]$ $e_i \in C_1$ (*ПриращениеУглаВектораКасательной*(e_i, e_{i+1}) ≤ 0);
<*Усл3*> ::= $\exists_{=2} U$ (U *is* <*Узел*> $\wedge$ $U \cap C_2 \neq \varnothing$).

<*Карман*> ::= <2*СвязанныхКонтура*>: <*Усл*1>, <*Усл*2>,
где <*Усл*1> ::= $\exists_{=2} U$ (U *is* <*Узел*> $\wedge$ $U \cap C_1 \neq \varnothing$);
<*Усл*2> ::= $\exists_{=2} U$ (U *is* <*Узел*> $\wedge$ $U \cap C_2 \neq \varnothing$).

Описание ТКЭ выполнено с помощью аппарата формальных грамматик и первопорядковой логики предикатов.

На этапе определения локальных максимумов было получено 89 кластерообразующих точек. В результате рекурсивного поиска границ кластеров была получена 51 группа деталей. Общее время работы алгоритма с учетом загрузки информации из базы данных составило 8 секунд на персональном компьютере.

На рис. 4 показана характеристика полученных кластеров. В значимые кластеры (размером от 2 шт.), представляющие пользу для последующей реализации групповой технологии, было отнесено 211 деталей (92.5%).

На рис. 5 показаны некоторые из полученных кластеров.

С точки зрения эксперта-технолога, в значимых кластерах неверно классифицированными оказались 13 деталей, и еще 3 детали, попавшие в кластеры-синглтоны, следовало отнести в значи-

мые кластеры. В целом, точность кластеризации составила 0.93.

Некоторые ошибки классификации были обусловлены тем, что не всегда удавалось правильно извлечь признаки геометрической формы из чертежей или используемый набор характеристик был недостаточен. Требуется усовершенствовать методы автоматического анализа чертежей, ввести новые признаки, учитывающие относительное расположение структурных элементов.

Некоторые кластеры (7 шт.) требуют повторной (иерархической) кластеризации – это удобнее для разработки групповых технологий. Чем меньше размер группы деталей, тем проще технологу с ней работать.

6. ЗАКЛЮЧЕНИЕ

Таким образом, предлагаемый алгоритм кластеризации множества деталей по чертежам на основе N-мерной гистограммы отличается от известных способом выборки соседних векторов, который состоит в пересчете координат из N-мерного пространства в одномерное и обратно и сравнении с координатами соседних векторов. Алгоритм осуществляет поиск вершин кластеров за один проход. Основным достоинством алгоритма является отсутствие необходимости создания множества локальных списков или графов соседних векторов, благодаря чему повышается эффективность алгоритма кластеризации. Также следует отметить, что алгоритм осуществляет кластеризацию деталей в терминах, обычно употребляемых технологами, за счет механизма перевода количественных значений признаков в качественные.

Оперативная (например, в ходе планирования позаказного многономенклатурного производства) кластеризация деталей по геометрическим, техническим, технологическим, производственным параметрам позволяет повысить степень унификации технологических процессов и за счет этого оптимизировать план производства.

7. БЛАГОДАРНОСТИ

Исследование алгоритма выполнено Касимовым Д.Р. за счет гранта Российского научного фонда (проект № 18-71-00109).

СПИСОК ЛИТЕРАТУРЫ

1. *Halevi G.* Expectations and Disappointments of Industrial Innovations. Lecture Notes in Management and Industrial Engineering. Springer, 2017. 131 p.
2. *Barladian B.Kh., Voloboy A.G., Galaktionov V.A., Shapiro L.Z.* Integration of Realistic Computer Graphics into Computer-Aided Design and Product Lifecycle Management Systems // Programming and Computer Software. 2018. V. 44. Is. 4. P. 225–232.
3. *MacQueen J.* Some methods for classification and analysis of multivariate observations // In Proc. 5th Berkeley Symp. on Math. Statistics and Probability. 1967. P. 281–297.
4. *Sidorova V.S.* Histogram Hierarchical Algorithm and the Reduction of the Dimensionality of the Spectral Features Space // Zh. Sib. Fed. Univ., Ser. Tekhn. Tekhnol. 2017. V. 10. Is. 6. P. 714–722.
5. *Narendra P.M., Goldberg M.* A non-parametric clustering scheme for LANDSAT // Pattern Recognition. 1977. V. 9. Is. 4. P. 207–215.
6. *Kharinov M.V.* Pixel clustering for color image segmentation // Programming and Computer Software. 2015. V. 41. Is. 5. P. 258–266.
7. *Ward J.H.* Hierarchical grouping to optimize an objective function // Journal of the American Statistical Association. 1963. V. 58. Is. 301. P. 236–244.
8. *Hodashinsky I.A., Mekh M.A.* Fuzzy classifier design using harmonic search methods // Programming and Computer Software. 2017. V. 43. Is. 1. P. 37–46.
9. *Wang L., Yang R., Xu Y., Niu Q., Pardalos P.M., Fei M.* An improved adaptive binary harmony search algorithm // Information Sciences. 2013. V. 232. P. 58–87.
10. *Bai S., Liu Z., Bai X.* Co-spectral for robust shape clustering // Pattern Recognition Letters. 2016. V. 83. P. 3. P. 388–394.
11. *Shen W., Wang Y., Bai X., Wang H., Latecki L.J.* Shape clustering: Common structure discovery // Pattern Recognition. 2013. V. 46. Is. 2. P. 539–550.
12. *Barton J., Love D.* Retrieving designs from a sketch using an automated GT coding and classification system // Production Planning & Control. 2005. V. 16. Is. 8. P. 763–773.
13. CADFind. URL: http://www.cadfind3d.com/
14. *Liu R., Wang Y., Baba T., Masumoto D.* Shape detection from line drawings with local neighborhood structure // Pattern Recognition. 2010. V. 43. Is. 5. P. 1907–1916.
15. *Fonseca M.J., Ferreira A., Jorge J.A.* Sketch-Based Retrieval of Vector Drawings // Sketch-based Interfaces and Modeling. 2011. V. 12. P. 181–204.
16. *Knuth D.* The Art of Computer Programming. Volume 2: Seminumerical Algorithms. 3-rd Ed. Massachusetts: Addison–Wesley, 1997. 762 p.
17. *Kasimov D.R., Kuchuganov A.V., Kuchuganov V.N.* Vectorization of Raster Mechanical Drawings on the Base of Ternary Segmentation and Soft Computing // Programming and Computer Software. 2017. V. 43. Is. 6. P. 337–344.
18. *Kasimov D.R., Kuchuganov A.V., Kuchuganov V.N.* Individual Strategies in the Tasks of Graphical Retrieval of Technical Drawings // Journal of Visual Languages and Computing. 2015. V. 28. P. 134–146.

ПРОГРАММИРОВАНИЕ, 2020, № 1, с. 39–53

ПАРАЛЛЕЛЬНОЕ И РАСПРЕДЕЛЕННОЕ ПРОГРАММИРОВАНИЕ

УДК 004.42

БЫСТРЫЙ АЛГОРИТМ КЛАССИФИКАЦИИ СЕЙСМИЧЕСКИХ СОБЫТИЙ НА БАЗЕ РАСПРЕДЕЛЕННЫХ ВЫЧИСЛЕНИЙ APACHE SPARK

© 2020 г. С. Е. Попов[a],*, Р. Ю. Замараев[a],**

[a] *Федеральное государственное бюджетное учреждение науки Институт вычислительных технологий Сибирского отделения Российской академии наук*
630090 Новосибирск, пр. Академика Лаврентьева, 6, Россия

**E-mail: popov@ict.sbras.ru*

***E-mail:zamaraev@ict.sbras.ru*

Поступила в редакцию 14.05.2019 г.
После доработки 19.08.2019 г.
Принята к публикации 19.08.2019 г.

В статье описаны ключевые моменты процесса разработки программной реализации алгоритма быстрой автоматической классификации сейсмических сигналов на основе диагностических шаблонов. Подробно описан процесс адаптации и интеграции данной реализации в систему распределенных вычислений Apache Spark. Представлено программное решение для предварительной обработки сигнала и оптимизации математической модели для параллельных вычислений с использованием транслируемых переменных. Проведены тесты производительности алгоритма классификации для набора суточных сигналов. Достигнуто десятикратное уменьшение времени работы алгоритма в контексте массивно-параллельного исполнения расчетных заданий по сравнению с последовательной обработкой.

DOI: 10.31857/S0132347420010057

1. ВВЕДЕНИЕ

Региональный мониторинг и анализ региональной геодинамической ситуации характеризуется сложностью решаемых на этом направлении задач. Причина в том, что наряду с мощными возмущениями из известных очаговых зон приходится анализировать и классифицировать разнородный поток событий, среди которых промышленные взрывы различной мощности и глубины заложения, региональные и местные сейсмические события. В горнопромышленных регионах России (Кемеровской области, Красноярском крае и др.) функционирует большое количество предприятий, регулярно проводящих массивные взрывные работы. В общей сложности за год может регистрироваться более 2.5 тысячи сейсмических событий со схожей магнитудой (1.5–2.5 балла), характерной как для региональных землетрясений, так и для типичных (по технологии и мощности) взрывов, например, на угольных разрезах или глубоких карьерах. Необходимо также учитывать разветвленную сеть сейсмостанций (например, по Красноярскому краю – 10, по Кемеровской области 7), записывающих непрерывный поток данных с частотой в среднем 100 отсчетов (замеров) каждые 10 мс минимум по трем каналам измерений в каждой. Таким образом накапливаются огромные массивы информации, где только для одной станции недельный пул суточных записей составляет около 150 Мб (≈174 млн отсчетов), с нескольких – более 1 Гб (1 млрд отсчетов). Учитывая данный фактор, процесс детектирования и классификации различных возмущений в наборе даже суточных записей с 4–6 станций требует значительных вычислительных ресурсов и затрат времени. Это же утверждение косвенно подтверждается анализом большинства исследований в области обработки сейсмических сигналов [1–17]. В этих публикациях описан ряд работоспособных подходов, которые опираются на различные комбинации процедур фильтрации, автокорреляционные методы, спектрального и вейвлет-анализа, интегрируемые с аппаратом искусственных нейронных сетей и кластерным анализом. В большинстве таких исследований, все апробации алгоритмов на реальных сигналах осуществляются на малых таймфреймах (временной промежуток или отрезок) размером не более 60–80 секунд. Рассматриваются отрезки сигнала с априори достоверной информацией о присутствии на них существенных (детектируемых) возмущений. Время работы представленных алгоритмов на полном сигнале (суточной записи) не приводится. Отдельно следует отметить работу [16], в которой авторы используют алгоритм (Fingerprint And Similarity

Thresholding (FAST)) для детектирования природных землетрясений и проводят его тестирование на реальных сигналах недельного пула записей. В работе показано время исполнения около 1 часа 36 минут против 9 дней автокорреляционного метода (данные взяты с одной станции). Таким образом при анализе даже 2–3 недельных таймфреймов с нескольких сейсмопостов время работы может увеличиваться до 1 дня, а с использованием другим методов и до месяца.

Рассматривая проблему производительности алгоритмов с позиций повышения эффективности информационно-аналитического обеспечения процессов регионального мониторинга можно обозначить основные требования: стабильное поступление массивов актуальных сейсмических данных; их оперативная обработка; анализ и выработка экспертных заключений с использованием быстрых алгоритмов. В мировой практике насчитывается большое количество программных решений, реализующих различные функции обработки и анализа сейсмической информации. Среди средств сбора, хранения и доступа к сейсмической информации крупнейшим является IRIS DMC [18]. Среди средств обработки и анализа сейсмических данных можно выделить программное обеспечение для обнаружения волновых возмущений сигнала [19–21], для интерактивного моделирования сейсмического волнового поля и проверки геологических гипотез при классификации сейсмических данных [22], а также программные комплексы для решения обратной геофизической задачи, построенные на базе открытых GRID-систем [23–25].

Однако функциональные возможности существующих программных решений не поддерживают классификацию сейсмических событий, основанную на обработке множества реальных суточных записей, полученных с разных станций наблюдений. Обработка данных ведется в ручном режиме, с последовательной загрузкой файлов, и выделением мелких таймфреймов интересующего события. Интегрированные в такие программные средства алгоритмы классификации не поддерживают запуск в параллельном режиме обработки полного временного отрезка сигнала. Все это приводит к существенной деградации производительности. В большинстве случаев программные решения представляют собой статические приложения, ориентированные на работу со специализированной аппаратной частью. Данный факт значительно сужает возможности анализа, поиска и подтверждения характера сейсмопроявлений на основе информации, получаемой одновременно из различных источников их фиксации.

Учитывая вышеизложенное, возникает актуальная задача разработки программного обеспечения для классификации сейсмических событий, поддерживающего высокопроизводительную обработку больших массивов данных в вычислительных средах на базе массивно-параллельного исполнения заданий.

2. МАТЕМАТИЧЕСКАЯ МОДЕЛЬ АЛГОРИТМА КЛАССИФИКАЦИИ

Алгоритм классификации является оригинальной разработкой авторов [27, 28]. В текущей версии алгоритма классификация на базе корреляционной функции расширена использованием функций статистических расстояний (дистанций). В связи с последним обстоятельством значительно улучшилась разрешающая способность алгоритма (количество верных классификаций), подтвержденная протоколами и оперативными донесениями Кемеровской службы мониторинга геофизической обстановки.

2.1. Исходные данные

Источником исходных данных для алгоритма служит сейсмический сигнал в формате miniSEED, состоящий из трех каналов в виде отдельных файлов (например, EHE, EHN, EHZ). Суточная запись с каждого канала в среднем содержит около 8.5 млн отсчетов (замеров с датчиков) и время каждого отсчета, т.е. (24 часа) × (3600 сек в часе) × (100 количество отсчетов в секунде, **sample_rate**). Начальные записи в каналах могут быть сдвинуты относительно начала суток (00:00:00). Для их синхронизации выбирается самый поздний по времени начальный отсчет (по максимальному времени от начала), и от этого времени извлекаются все значения каждого канала. Далее, выбирается минимальная длина из получившихся массивов (по минимальному времени от конца сигнала), оставшиеся массивы “обрезаются справа” до этой длины. Таким образом формируется исходный массив классифицируемого сигнала (1) со значениями типа Double:

$$CH = \{ch_{ij}, i \in [0, L_{ch}] \in \mathrm{Z}, j \in [0,2] \in \mathrm{Z}\}, \qquad (1)$$

где L_{ch} – длина массива данных для каждого синхронизированного канала (в среднем 8.3–8.5 млн элементов), j – номер канала.

2.2. Предварительная обработка сигнала

Алгоритм позволяет анализировать полные трехкомпонентные суточные сигналы. На вход алгоритму подается сформированная матрица (1) ($CH = \{ch_{ij}, i \in [0, L_{ch}] \in \mathrm{Z}, j \in [0,2] \in \mathrm{Z}\}$. Задается скользящее окно размером в $m = 6145$ отсчетов, с шагом сдвига $step = 100$ отсчетов. На каждом шаге формируется сейсмограмма в виде матрицы $X = \{x_{ij}, \ i \in [0,m] \in \mathrm{Z}, j \in [0,2] \in \mathrm{Z}\}$, содержащая часть сигнала CH (1). Алгоритм определяет (клас-

сифицирует) тип сейсмограммы согласно шаблонам, следующим образом:

Шаг 1. Компоненты матрицы $X = \{x_{ij}\}$ заменяем квадратами размахов $sw_{i,j}$, что обеспечивает неотрицательность значений для дальнейших вычислений

$$sw_{i,j} = \left(x_{i,j} - x_{i+1,j}\right)^2, \quad i \in [0, m-1], \quad j \in [0,2]. \quad (2)$$

Шаг 2. Вычисляется матрица весов (3) и, затем, матрица энтропий (4)

$$q_{i,j} = \frac{sw_{i,j}}{\sum_{i=0}^{m-1} sw_{i,j}}, \quad i \in [0, m-1], \quad j \in [0,2]. \quad (3)$$

$$E_{i,j} = -q_{i,j} \ln\left(q_{i,j}\right), \quad i \in [0, m-1], \quad j \in [0,2]. \quad (4)$$

Энтропийная модель дискретного сигнала (3–4) обладает свойствами Шенноновской энтропии выборки [29]. Модель обеспечивает аддитивность элементов, относящихся к одному сигналу и, главное, аддитивность элементов из различных сигналов в синхронных отсчетах.

Шаг 3. Вычисляется вектор обобщенной информации по трем каналам измерений

$$H_i = E_{i,0} + E_{i,1} + E_{i,2}, \quad i \in [0, m-1]. \quad (5)$$

Шаг 4. Строится характеристическая функция в рачетном окне (6). Данный процесс называется аккумулированием сигнала

$$C_i^s = \sum_{l=0}^{i} H_l, \quad i \in [0, m-1]. \quad (6)$$

За счет аккумулирования три компоненты сигнала приводятся к одномерной стационарной форме [30]. Стационарность модели (6) обеспечивает хорошую аппроксимацию по осредненным (сглаженным) данным (шаблонам с известными характеристиками).

2.3. Построение типовых шаблонов

Идея алгоритма классификации такова, что для построения шаблона можно использовать любое доступное количество сейсмограмм подтвержденных промышленных взрывов и региональных землетрясений. Усреднение шаблонов, естественно, проявляет особенности характеристических функций соответствующих классов событий, но стирает различия между событиями одного класса.

Для каждого достоверного события по приведенному выше алгоритму (1)–(6) строятся характеристические функции. Если достоверных событий достаточно, то для совокупностей взрывов и землетрясений раздельно вычисляется по три шаблона: средний шаблон и два соответствующих границам $S = 0.5\sigma$, где σ – среднеквадратическое отклонение, вычисляемое как

$$\sigma_i = \sqrt{\frac{1}{U_1 \text{ или } U_2} \sum_{j=1}^{U_1 \text{ или } U_2} (C_{i,j} - \mu_i)^2}, \mu_i = \frac{1}{U_1 \text{ или } U_2} \sum_{j=1}^{U_1 \text{ или } U_2} C_{i,j},$$

где U_1, U_2 – необходимое количество достоверных сейсмограмм промышленных взрывов и региональных землетрясений, соответственно. В среднем для одной сейсмостанции выбирается $U_1 = 30, U_2 = 19$.

Откуда для совокупностей взрывов и землетрясений получаются по три шаблона: средний и две его границы: для взрывов Blast, Blast ± S (B, B ± S), для землетрясений EarthQuake, EarthQuake ± S (E, E ± S).

Исследования авторов показали, что для станций сейсмических наблюдений, достаточно удаленных от зашумленных промышленных зон, характерное время прохождения и эффективного затухания сейсмического возмущения от взрывов находится в диапазоне 50...75 сек. Таким образом, длину шаблона (расчетное окно) в среднем можно принять равным $m = 6145$ отсчетов или 61.45 секунды при частоте дискретизации сигнала 100 Гц.

Еще одна особенность алгоритма – абстрактные шаблоны. Они получаются с помощью функции $f(t) = A\left(\frac{t}{Th}\right)^h \exp\left(h - \frac{t}{Th}\right) + \varepsilon(t)$: путем вариации значений параметров T и h в (6) при $t = 0, \ldots, m$ формируется набор одномодальных огибающих для вектора обобщенной информации **H** (5). По формуле (6) из этих огибающих получается набор абстрактных характеристических функций. Они изображают последовательное прохождение одномодального сейсмического возмущения через расчетное окно, и для них введены следующие обозначения:

- три шаблона на входе в расчетное окно: WaveFront-I (WF-I), WaveFront-II (WF-II) и WaveFront-III (WF-III);
- три шаблона на выходе из расчетного окна: WaveRear-I (WR-I), WaveRear-II (WR-II), WaveRear-III (WR-III);
- три шаблона в середине расчетного окна: WaveMiddle (WM), WaveLeft (WL) и WaveRight (WR).

Таблица 1. Статистические расстояния

Название расстояния	Формула для расчета	Формула для расчета с весовым коэффициентом
Bray-Curtis	$D_{0,j} = \frac{\sum_{i=0}^{m-1} \lvert S_{i,16} - S_{i,j} \rvert}{\sum_{i=0}^{m-1} \lvert S_{i,16} + S_{i,j} \rvert}$	–
Canberra	$D_{1,j} = \sum_{i=0}^{m-1} \frac{\lvert S_{i,16} - S_{i,j} \rvert}{\lvert S_{i,16} \rvert + \lvert S_{i,j} \rvert}$	$D_{2,j} = \sum_{i=0}^{2(m-1)/3} w_i \frac{\lvert S_{i,16} - S_{i,j} \rvert}{\lvert S_{i,16} \rvert + \lvert S_{i,j} \rvert}$
CityBlock	$D_{3,j} = \sum_{i=0}^{m-1} \lvert S_{i,16} - S_{i,j} \rvert$	–
“Корреляция (нормированная)”	$D_{4,j} = 1 - cov/(\sigma_1 \sigma_2),$ $cov = \frac{\sum_{i=0}^{m-1} S_{i,16} S_{i,j}}{m-1} - \frac{\sum_{i=0}^{m-1} S_{i,16} \sum_{0}^{m-1} S_{i,j}}{(m-1)^2}$ $\sigma_1 = \sqrt{\frac{\sum_{i=0}^{m-1} S_{i,16}^2}{m-1} - \frac{\left(\sum_{i=0}^{m-1} S_{i,16}\right)^2}{(m-1)^2}}$ $\sigma_2 = \sqrt{\frac{\sum_{i=0}^{m-1} S_{i,j}^2}{m-1} - \frac{\left(\sum_{i=0}^{m-1} S_{i,j}\right)^2}{(m-1)^2}}$	–
“Евклидово”	$D_{5,j} = \lVert S_{16} - S_j \rVert_2$	$D_{6,j} = \sqrt{\sum_{i=0}^{2(m-1)/3} w_i \left(S_{i,16} - S_{i,j}\right)^2}$
“Евклидово второй степени”	$D_{7,j} = \lVert S_{16} - S_j \rVert$	$D_{8,j} = \sum_{i=0}^{2(m-1)/3} w_i \left(S_{i,16} - S_{i,j}\right)^2$
“Минковского третьей степени”	$D_{9,j} = \lVert S_{16} - S_j \rVert_3$	$D_{10,j} = \sqrt[3]{\sum_{i=0}^{2(m-1)/3} w_i \left(S_{i,16} - S_{i,j}\right)^3}$
“Косинусное”	$D_{11,j} = 1 - \frac{S_{16} \cdot S_j}{\lVert S_{16} \rVert_2 \lVert S_j \rVert_2}$	–

Примечание. Прочерк означает отсутствие аналогичного расстояния с весовым коэффициентом.

Дополнительно вводится абстрактный шаблон WhiteNoise (WN), изображающий сейсмический фон и возможные мультимодальные малоамплитудные возмущения. Он получен как характеристическая функция одноуровневого сигнала $f(t) = 1$.

В текущей версии алгоритма используется массив из 16 шаблонов типа Double, представленная как C_{ij}^t, $i \in [0,6144] \in \mathrm{Z}$, $j \in [0,15] \in \mathrm{Z}$.

2.4. Построение диагностической матрицы

Согласно (6), все шаблоны находятся в одном метрическом пространстве. Полагая шаблоны признаками, а отсчеты объектами (независимыми наблюдениями), можем дополнить их набор выборочной характеристической функцией и вычислить аналог диагностической матрицы по Байесу путем стандартизации в объектах по формуле (7).

Шаг 5. Справа к матрице $C_{ij}^t, i \in [0,6144] \in \mathrm{Z}$, $j \in [0,15] \in \mathrm{Z}$ добавляется вектор-столбец C^s, получается матрица $C_{ij}, i \in [0, m-1]$, $j \in [0, n]$, $n = 16$.

$$S_{i,j} = \frac{(C_{i,j} - \mu_i)}{\sigma_i}, \quad \text{где} \quad \mu_i = \frac{1}{n}\sum_{j=1}^{n} C_{i,j} \quad \text{и} \quad \sigma_i = \sqrt{\frac{1}{n}\sum_{j=1}^{n}\left(C_{i,j} - \mu_i\right)^2} \tag{7}$$

Теперь в матрице (7) можно оценивать подобие характеристической функции (6) каждому шаблону

Таблица 2. Пример заключения классификации для одного шага расчетного окна

№	Тип шаблона	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8	D_9	D_{10}	D_{11}	R	Заключение
1	WR-I	0	0	0	0	0	0	0	0	0	0	0	0	0	
2	WR-II	0	0	0	0	0	0	0	0	0	0	0	0	0	
3	WR-III	0	0	0	0	0	0	0	0	0	0	0	0	0	
4	WL	0	0	0	0	0	0	0	0	0	0	0	0	0	
5	B+S	0	0	0	0	0	0	0	0	0	0	0	0	0	
6	B	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**9**	**2**
7	B-S	0	0	0	0	0	0	0	0	0	0	0	0	0	
8	WM	0	0	0	0	0	0	0	0	0	0	0	0	0	
9	EQ+S	0	0	0	0	0	0	0	0	0	0	0	0	0	
10	EQ	0	0	1	0	0	0	0	1	0	0	1	0	3	
11	EQ-S	0	0	0	0	0	0	0	0	0	0	0	0	0	
12	WR	0	0	0	0	0	0	0	0	0	0	0	0	0	
13	WF-III	0	0	0	0	0	1	1	0	0	0	0	0	2	
14	WF-II	0	0	0	0	0	0	0	0	0	0	0	0	0	
15	WF-I	0	0	0	0	0	0	0	0	0	0	0	0	0	
16	WN	0	0	0	0	0	0	0	0	0	0	0	0	0	

из набора C_{ij}^t, как расстояние между двумя признаками (одномерными векторами). Для этого фиксируется $j = 16$, и для каждого $S_{i,j}$, $j \in [0,15]$ формируется пара с $S_{i,16}$, получается 16 пар.

Шаг 6. Для каждой пары рассчитываются значения расстояний по таблице 1.

Получается матрица расстояний

$$D = \{D_{k,j}, k \in [0,11], j \in [0,15]\}, \quad (8)$$

где весовой коэффициент

$$w_i = \begin{cases} 1, & i \in [0, 2(m-1)/3] \\ 0, & \text{в остальных случаях.} \end{cases}$$

Априорных суждений о преимуществах тех или иных дистанций не существует, поэтому в текущей версии алгоритма используются все, пригодные для номинальных признаков с различными вариациями [31].

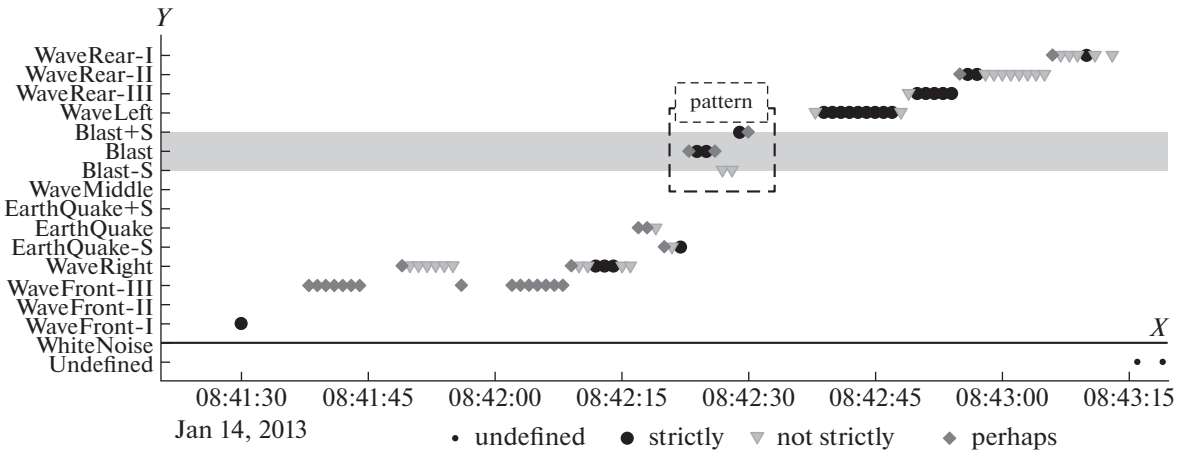


Рис. 1. Карта классификаций суточной записи сейсмического сигнала с выделенным паттерном промышленного взрыва.

```
"тип заключения": {"отсчет (№ сек.)": [...], "№ типа шаблона (табл. 2)": [...]}

      "strictly": {"x": [...,31345,31346,...], "y": [...,6,6,...]},
      "strictly": {"x": [...,31352,...],"y": [...,5,...]},
      "perhaps": {"x": [...,31353,...],"y": [...,5,...]},
      "not strictly": {"x": [...,31348,...],"y": [...,7,...]},
      "not strictly": {"x": [...,31349,...],"y": [...,7,...]},
```


Рис. 2. Фрагмент искомого паттерна, соответствующего промышленному взрыву, записанного в формате JSON.

2.5. Классификация

В алгоритме реализована простейшая система голосования, в которой каждая дистанция имеет один голос. Каждый голос отдается шаблону с минимальной дистанцией (8) до выборочной характеристической функции. Простое суммирование голосов у каждого шаблона определяет его простой рейтинг.

Шаг 7. Матрица D преобразуется следующим образом: для каждого $k \in [0,11]$

$$D_{k,j} = \begin{cases} 1, & D_{k,j} = \min(D_k), \\ 0, & \text{в остальных случаях.} \end{cases} \quad (9)$$

Шаг 8. Рассчитываются рейтинги

$$R_j = \sum_{k=0}^{11} D_{k,j} \quad (10)$$

и вычисляется максимум $\{R_j\}$

Шаг 9. Заключение классификации формируется по следующей схеме, назовем ее рейтинговым голосованием (Таблица 2):

а) если не существует единственного максимума, то заключение “**не определено (undefined = 0)**”;

б) если единственный максимум больше 10, то заключение “**строгое (strictly = 1)**” соответствие шаблону;

в) если единственный максимум больше 8, но меньше 11, то заключение “**нестрогое (not strictly = 2)**” соответствие шаблону;

г) если единственный максимум меньше 9, то заключение “**возможное (perhaps = 3)**” соответствие шаблону.

В таблице 2 показано, что на текущем шаге расчетного окна, форма сигнала длиной 6144 отсчета соответствует (не строго) шаблону Blast, т.к. исходя из условий классификатора (**Шаг. 9**) получаем: $8 < \max(R) < 11$ и максимум единственный. Соответственно на каждом шаге итерации равной 1 сек получаем заключение в виде номера итерации, равной количеству секунд от начала сигнала, и заключения, представленного парой: тип шаблона и номер заключения.

Сдвиг расчетного окна от начала сигнала в конец, позволяет детектировать и идентифицировать согласно шаблонам любые значимые возмущения суточного таймфрейма, формируя таким образом полную карту классификаций (рис. 1).

Для анализа и идентификации полученных классификаций на каждом сдвиге окна ищутся характерные для региональных землетрясений и промышленных взрывов паттерны (рис. 1). Паттерн – это последовательность заключений классификаций (таблица 1), точки на графике, расположенные в определенном порядке в расчетном окне. Для паттернов промвзрывов и региональных землетрясений общий порядок расположения и тип точек по оси X одинаков, и отличается только по оси Y. Для однозначной идентификации необходимо выполнение одновременно следующих условий:

- в интервале не менее 5 секунд для шаблона **Blast/Earthquake** обязательное наличие в паттерне одного или более заключений типа **strictly = 1**. Также допустимо включение нескольких заключений типа **not strictly = 2** и **perhaps = 3**;
- в этом же 5-ти секундном интервале необходимо обязательное наличие одного или более заключений типа **strictly = 1** и/или **not strictly = 2** и/или **perhaps = 3** для шаблонов **Blast/ Earthquake** $\pm$ **S**, соответственно.

Для автоматизации поиска паттернов карта классификаций записывается в виде файла JSON-формата (рис. 2, см. раздел **Адаптация алгоритма для распределенных вычислений в системе Apache Spark**).

3. АДАПТАЦИЯ АЛГОРИТМА ДЛЯ РАСПРЕДЕЛЕННЫХ ВЫЧИСЛЕНИЙ В СИСТЕМЕ APACHE SPARK

Анализ алгоритма классификации (шаги 1–9) показал независимость расчетов на каждой итерации сдвига расчетного окна. Этот факт означает, что заключение классификации получается, как единичное абстрактное значение одного из признаков (см. шаг 9, раздел 2**. Алгоритм классификации, 2.5 Классификация**). Следовательно, можно разделить всю суточную запись длинной L_{ch} на части (**partitions**) по количеству всех доступных ядер кластера, и вычислять модель (2)–(10)

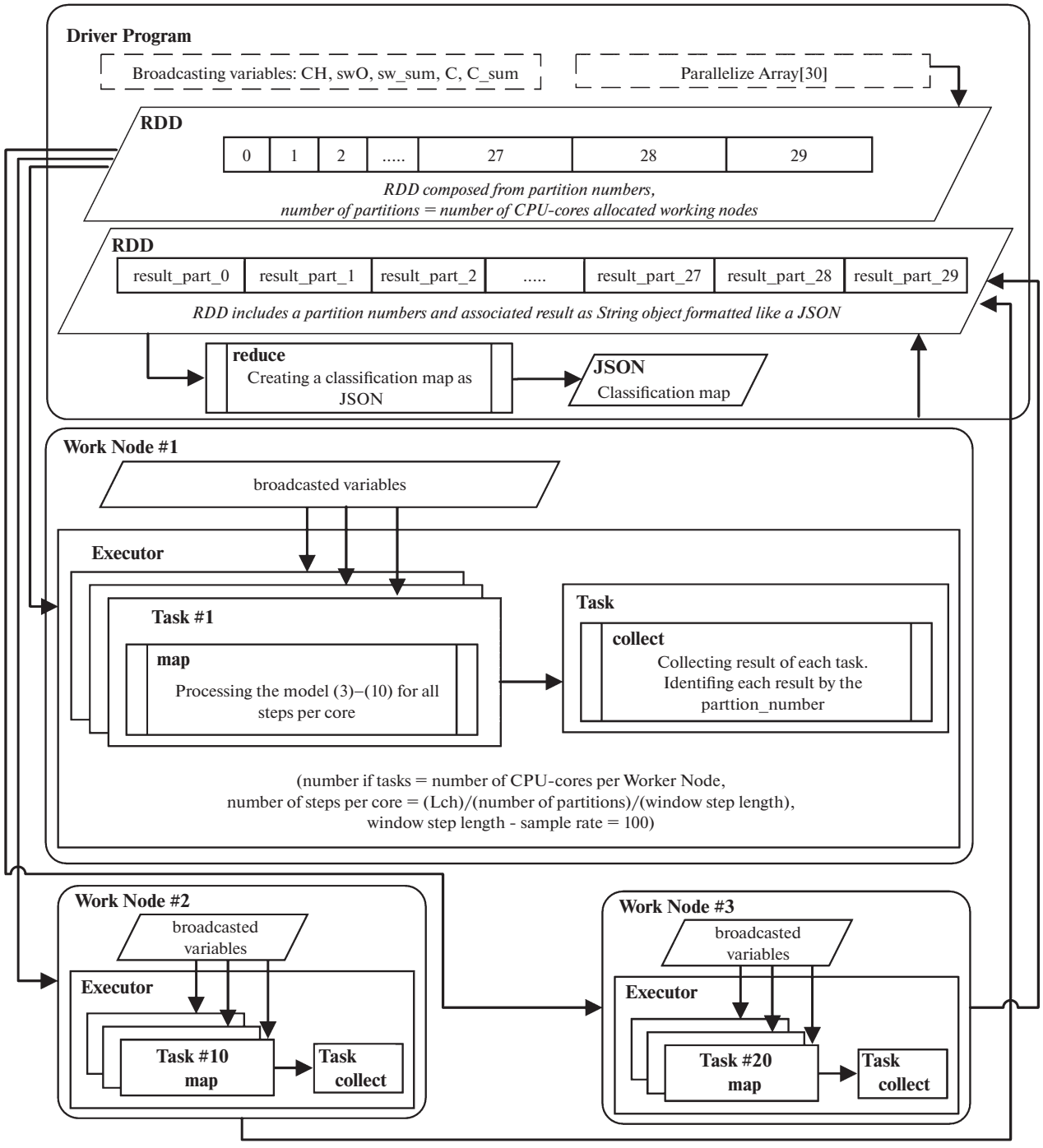


Рис. 3. Общая схема организации вычислений в системе Apache Spark для алгоритма классификации на трех физических узлах кластера с выделением по 10 вычислительных ядер на каждом.

параллельно (этап **Map)**. Затем после завершения всех заданий **Map**, запускать процесс сбора (**collect**) полученных результатов каждого из заданий (**Task**), присваивая им номер соответствующего **partition** (рис. 4). И уже на последнем этапе, на стороне управляющей программы (**Driver Program**) объединить полученные результаты, сортируя их по времени в начальном сигнале (карта классификаций) – этап **Reduce**. Таким образом возможно организовать вычисления по классической схеме **MapReduce** с промежуточным этапом **Collect** (рис. 3).

Карта классификации формируется на стороне управляющей программы после этапа **Reduce** в виде JSON-файла следующего вида (рис. 4), где поле **x** содержит номер шага расчетного окна или

(а)

Этап «Collect»

```
{
  "partition00": {
    "undefined": {...},
    "strictly": {
      "x": [
        721,
        722,
        723,
        ...
      ],
      "y": [
        3,
        4,
        6,
        ...
      ],
      "time": [
        "2013-01-14 00:12:04",
        "2013-01-14 00:12:05",
        "2013-01-14 00:12:13",
        ...
      ]
    },
    "notstriatly": {...},
    "perhaps": {...}
  },
  ...
  "partition29": {
    "strictly": {
      "x": [
        ...
        85432,
      ],
      "y": [
        ...
        12
      ],
      "time": [
        ...
       "2013-01-14 23:43:54"
      ]
    }
  }
}
```

(б)

Этап «Reduce»

```
{
  "underined": {...},
  "striotly": {
    "x": [
      721,
      722,
      723,
      ...
      85432
    ],
    "y": [
      3,
      4,
      6,
      ...
      12
    ],
    "time": [
      "2013-01-14 00:12:04",
      "2013-01-14 00:12:05",
      "2013-01-14 00:12:13",
      ...
      "2013-01-14 23:43:54"
    ]
  },
  "notstrictly": {...},
  "perhaps": {...},
  "channe11":"AN.BRCR.81.EHE.D.2013.014"
  "channe12": "AN.BRCR.81.EHN.D.2013.014",
  "channe13": "AN.BRCR.81.EHZ.D.2013.014",
  "signalStartTime": "2013-01-14 00:04:02",
  "signalEndTime": "2013-01-14 23:58:01"
}
```

Рис. 4. Карта классификаций суточного сейсмического сигнала в виде JSON-файла на этапах а) **Collect** и б) **Reduce**, соответственно.

количество секунд от начала сигнала, поле **y** – номер классифицированного шаблона (табл. 2) для текущего расчетного окна, поле **time –** время, соответствующее первому отсчету в расчетном окне. А также дополнительно указываются названия файлов каналов и начальное и конечное время после их синхронизации.

На первом этапе происходит предварительная подготовка данных. Для этого создается объект **RDD** (Resilient Distributed Datasets), содержащий номера разделов (**partition**), в соответствии с выделенным количеством вычислительных ядер кластера (**number_of_partitions**). Далее определяется количество шагов расчетного окна на каждое ядро (**steps_per_core**) (11). Один шаг расчетного окна – это сдвиг на одну секунду в суточной записи сигнала.

$$\text{steps\_per\_core} = \frac{L_{ch}}{(number\_of\_partitons)(sample\_rate)} \qquad (11)$$

Соответственно, каждое задание (**Task**) будет обрабатывать только часть массива CH. Индекс

```
{
  // Основной java-класс
  "className": "org.myapp.seismatica.classifiers.ClassificationProcessor",
  // Путь к файлу программы
  "file": "hdfs://cloudera-node04/user/jars/seismatica/seismatica-classifier-1.0.jar",
  "name": "Seismatica - Classifier",
  "args": [
    "DistanceClassifier", // Название классификатора
    chFilesArg, // Массив путей к файлам каналов
    templateFile, // Путь к файлу шаблонов
    "100", // Шаг сдвига окна
    "30" // Количество задач (исполняются параллельно)
  ],
  "conf": {
    "spark.executor.instances": "3", // Количество Executor-объектов (работают параллельно)
    "spark.task.cpus": "1", // Количество CPU на одно задание
    "spark.executor.cores": "10", // Количество задач на один Executor
    "spark.executor.memory": "4g", // Выделяемая память для одного Executor
    "spark.driver.memory": "2g", // Выделяемая память для управляющей программы
    "spark.driver.extraClassPath": "/mnt/hdfs/user/jars/seismatica/*", // Путь к java-классам
    "spark.executor.extraClassPath": "/mnt/hdfs/user/jars/seismatica/*" // Путь  к java-классам
  }
}
```

Рис. 5. Пример JSON-объекта в POST-запросе к сервису Apache Livy на удаленный запуск задания (Task).

начального элемента массива рассчитывается по формуле (12).

$$\text{start\_idx} = (steps\_per\_core)(part_i), \quad i = 0...numper\_of\_partitions - 1 \tag{12}$$

Весь массив CH представлен в виде специального объекта **Broadcast** фреймворка Spark API. **Broadcast** – это транслируемая переменная, хранящаяся в кэше на каждом узле (Worker Node) кластера. В отличии от обычной переменной, **Broadcast** не отправляется как копия на каждую задачу (Task), что позволяет эффективно работать с большими наборами неизменяемых данных, каковыми являются суточные записи сейсмического сигнала.

Использование транслируемых переменных позволило значительно оптимизировать представленный алгоритм. Для уменьшения времени работы программной реализации алгоритма и его адаптации к запуску в среде Apache Spark были проведены перечисленные далее действия.

1. Предварительно рассчитываются элементы $sw_{i,j}$ (2) для трех каналов суточной записи. Формируется массив $swO_{ij}, i \in [0, L_{ch} - 1]$. Заранее вычисляются $\sum_{i=1}^{m-1} sw_{i,j}$ (2) для каждого шага сдвига. Получается массив

$$sw\_sums_{j,s} = sw\_sums_{j,s-1} \pm \sum_{l=1}^{100} swO_{100s\pm l,j}, \quad s \in \left[1, \frac{L_{ch}}{100}\right], \quad \text{где}$$

$$sw\_sums_{j,0} = \sum_{i=0}^{m-1} sw_{i,j}.$$

Массивы swO и sw_sums объявляются транслируемыми переменными для всех заданий в среде Apache Spark, и передаются при помощи специального объекта **Broadcast**. Данная оптимизации позволяет существенно сократить время расчета формул (2) и (3), т.к. на каждом сдвиге окна вычисляются суммы только предыдущих и последующих 100 элементов массива sw.

2. Для массива C^t предварительно рассчитывается сумма $CSum_i = \sum_{j=1}^{n} C_{i,j}^t$. $CSum_i$ и аналогично объявляется транслируемой переменной. Тогда на каждом шаге выражение $\mu_i = \frac{1}{n}\sum_{j=1}^{n} C_{i,j}$ в формуле (7) можно заменить на следующее $\mu_i = \frac{1}{n}(C|i,16 + CSum_i)$, что также позволяет сократить количество операций с суммой элементов.

3. Используемые алгоритмы нахождения расстояний согласно [14] в своих расчетах содержат повторяющиеся выражения. Например, расстояние Bray-Curtis будет содержать выражение $S_{i,16} - S_{i,j}$, которое также есть в Canberra, или $\Sigma\left|S_{i,16} - S_{i,j}\right|$, $j \in [0,15]$ в City Block. Расчет корреляции через ковариацию, позволит получить выражения: $\Sigma S_{i,16}^2$, $\Sigma S_{i,j}^2$, $\Sigma S_{i,16} S_{i,j}$, которые присутствуют в расчете евклидова и косинусного расстояний. Учитывая, что весовой коэффициент принимает значение 1 при двух третьих от общего количества итераций, а

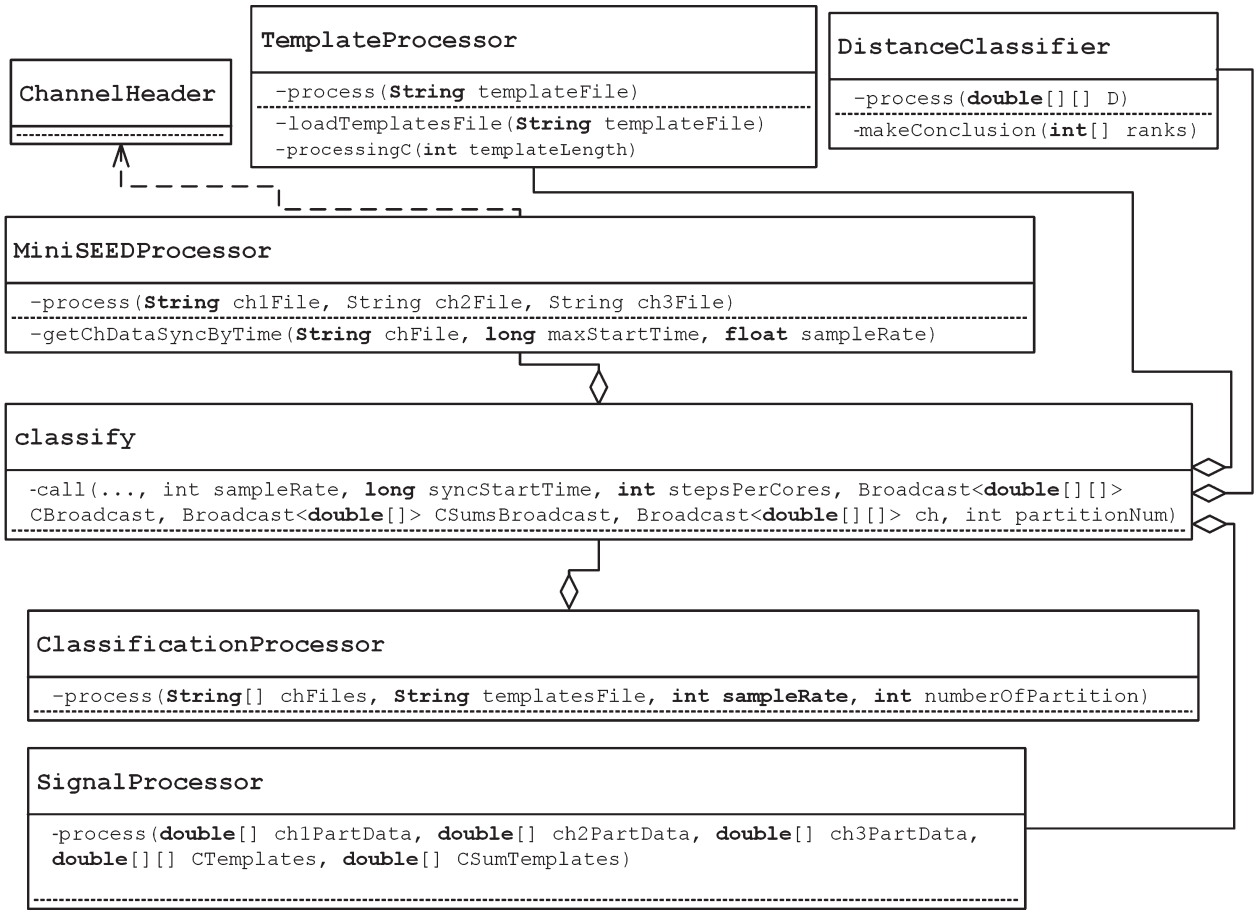


Рис. 6. Диаграмма классов программной библиотеки алгоритма.

остальные значения равны нулю, то при расчете сумм в соответствующих расстояниях без весового коэффициента, необходимо зафиксировать значение суммы на номере итерации равном $2(m-1)/3$, и использовать ее при получении значения расстояния с весовым коэффициентом. Т.е. если евклидово расстояние есть $\Sigma S_{i,16}^2 + \Sigma S_{i,j}^2 - 2\Sigma S_{i,16} S_{i,j}$, $i \in [0, m-1]$, то евклидово с весовым коэффициентом будет вычисляться также, только для $i \in [0, 2(m-1)/3]$.

Такого рода оптимизации позволяют значительно сократить время работы алгоритма, т.к. при каждом сдвиге приходится выполнять одни и те же вычислительные операции на одних и тех же наборах данных по нескольку раз. Учитывая количество шагов (в среднем 84000) и размер окна ($m-1=6144$ отсчета) благодаря предложенным оптимизациям выигрыш производительности может быть существенным.

На втором этапе каждое задание независимо рассчитывает модель (3)–(10), получает часть результата классификаций и идентифицирует его как **partition##**, где ## – номер раздела (рис. 4а), после чего запускается процедура объединения (рис. 4б). Полученный результат сохраняется в JSON-файл.

На третьем этапе происходит анализ карты классификаций, где выделяются характерные паттерны для типа событий. Строится таблица классифицированных сейсмических событий.

4. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

Алгоритм реализован в виде программной библиотеки на языке Java (jar-файл) (см. раздел **Исходный код**). Вычислительное ядро использует фреймворк Apache Spark API (Java) и менеджер ресурсов Apache YARN. Результаты сохраняются в систему HDFS. Для удобства работы с операциями ввода/вывода, файловая система HDFS подмонтирована как обычная папка в системе Linux, используя пакет hadoop-hdfs-fuse. Запуск расчетов происходит через систему Apache Livy с использованием файла параметров (рис. 5).

Программная библиотека содержит Java-классы, отвечающие непосредственно за расчетную часть алгоритма, и дополнительные классы, им-

Таблица 3. Тест производительности программной реализации представленного алгоритма классификации (время работы)

	Java Spark API	Python Spark API	Matlab	Java
	Программный код запускается параллельно на 30 вычислительных ядрах, с разбивкой на partitions		Программный код запускается последовательно, без применения Apache Spark API	
	Суточная запись			
Время работы (сек.)	17	24	3574	801
Время работы (сек.)	Недельный таймфрейм			
	154	183	25145	5599

Примечание. Суточная запись, синхронизированная по каналам в среднем, составляла 8355839 отчетов с интервалом в 10 мс или 83558 сдвигов. Аппаратное обеспечение: 3 сервера (AMD Ryzen 1700 (8+8 cores (Simultaneous Multi-Threading)) 3.2 GHz, 32Gb RAM, 1Gb/s скорость передачи данных между серверами). Локальный тест проводился на одном сервере.

плементирующие методы предварительной обработки сейсмического сигнала и работы с объектами Apache Spark API (рис. 6).

ClassificationProcessor основной класс, который отвечает за запуск процесса классификации. Содержит подкласс **classify** наследующий объект

```
// Обработчик шаблона
TemplateProcessor templateProcessor = new TemplateProcessor();
templateProcessor.process(templatesFile);
// Обработчик каналов
MiniSEEDProcessor miniSEEDProcessor = new MiniSEEDProcessor(...);
miniSEEDProcessor.process(chFiles[0], chFiles[1], chFiles[2], true);
// Настройка контекста запуска
SparkConf sparkConf = new SparkConf();
JavaSparkContext sc = new JavaSparkContext(sparkConf);
// Размещение общедоступных данных
Broadcast<double[][]> CBroadcast = sc.broadcast(templateProcessor.C);
...
// Запуск задания
List<String> classificationMapParts = sc.parallelize[IntStream.range(0,partitionCount)
         .boxed().collect(Collectors.toList()), partitionCount)
         .map(new classify(...,..., CBroadcast,...
        )).collect();
...
// Имплементация объекта Function для метода map (параллельно для каждого задания)
class classify implements Fcnction<Integer, String> {
...
public classify(...,...,Broadcast<double[][]> CBroadcast,...) {
        this.CBroadcast = CBroadcast;}

    @Override
    public String call(Integer core) {

        SignalProcessor signalProcessor = new SignalProcessor(...);
        DistanceClassifier distanceClassifier = new DistanceClassifier(...);
        ...
        // Доступ к общим данным внутри задания SparkContext
        double[][] C = CBroadcast.value();
        ...
        ...
        // Расчет алгоритма классификации
        double[][] D = signalProcessor.process(chlRawData, ch2RawBata, ch3RawData, C, CSums);
        conclusionResult = distanceClassifier.process(D);
        ...
        return conclcsionResult // Результат в виде JSON-строки
...
```

Рис. 7. Фрагмент кода настройки контекста Spark и запуска расчетного задания класса **ClassificationProcessor**.

```
double[][] D = signalProcessor.process(ch1RawData, ch2RawData, ch3RawData, C, CSums);
conclusionResult = distanceClassifier.process(D);
if (conclcaionResult[0] != -1) {
    date.setTime(syncStartTime + (core * stepsPerCores * 1000 + i * 1000)};
    // Формирование строки формирование строки в формате JSON для каждого из заключений
    ...
    if (conclusionResult[0] == 1) {
        strictlyX += (globalStartPosition + i) + ",";
        strictlyY += conclusionResult[1] + ",";
        strictlyTime += "\"" + simpleDateFormat.format(date).toString() + "\",";
    }
    ...
    if (conclusionResult[0] == 3) {
        perhapsX +- (globalStartPosition + i) + ",";
        perhapsY += conclusionResult[1] + ",";
        perhapsTime += "\"" + simpleDateFormat.format(date).toString() + "\",";
    }
}
...

// Конкантенация строк заключений
result = "\"undefined\":{\"x\":[" + undefinedX + "],\"y\":[" + undefinedY +
"],\"times\":[" + undefinedTime + "]},"
+ "\"strictly\":{\"x\":[" + strictlyX + "],\"y\":[" + strictlyY + "],\"times\":[" +
strictlyTime + "]},"
+ "\"notstrictly\":{\"x\":[" + notStrictlyX + "],\"y\":[" + notStrictlyY + "],\"times\":["
+ notStrictlyTime + "]},"
+ "\"perhaps\":{\"x\":[" + perhapsX + "],\"y\":[" + perhapsY + "],\"times\":[" +
perhapsTime + "]},";

return "\"partition" + String.format("%03d", core} + "\":" + "{" + result + "}";
```

Рис. 8. Фрагмент кода формирования объекта типа String, содержащего строку в формате JSON.

Function Spark API для передачи его в функцию **map**. Подкласс **classify** реализует программный алгоритм классификации (классы **SignalProcessor** и **DistanceClassifier**), адаптированный для работы в распределенном режиме на узлах кластера. Класс **ClassificationProcessor** обеспечивает настройку среды исполнения (**Executor**) заданий посредством объекта **SparkContext**. В **ClassificationProcessor** реализована процедура размещения транслируемых переменных (**Broadcast**), содержащих предварительно рассчитанные данные (см. раздел **Адаптация алгоритма**) в классах **TemplateProcessor** и **MiniSEEDProcessor** (рис. 7). Данные переменные доступны со всех узлов кластера в общей памяти текущего контекстного объекта.

Данные шаблонов хранятся в HDFS в CSV-файле. Класс **TemplateProcessor** поддерживает методы чтения, обработки данных CSV и построение массива C_{ij}^{t} и $\sum_{j=1}^{n} C_{i,j}$ для каждого i-го отсчета, для добавления в пул транслируемых переменных (**Адаптация алгоритма**).

Класс **MiniSEEDProcessor** содержит методы работы с файлами miniSEED-формата при помощи библиотеки iris-WS.jar. Она позволяет открывать, декодировать и считывать данные из файлов каналов сейсмических записей. **MiniSEEDProcessor** реализует процедуру синхронизации каналов по времени и формирования переменной **Broadcast** для матрицы CH (1). Так же в данном классе рассчитываются вспомогательные данные: длина сигнала и метаинформация по каждому каналу (название, частота дискретизации, начальное/конечное время записи).

Результатом работы метода **classify** (рис. 8) является JSON-файл (рис. 4а), содержащий одну из частей **partition##**.

После завершения всех заданий (Task) на стороне управляющей программы (Driver Program), происходит конечное объединение результатов в один объект типа String, имеющий вид (рис. 4б). Данный объект записывается в JSON-файл в файловую систему HDFS (рис. 9).

5. ТЕСТ ПО ОЦЕНКЕ ПРОИЗВОДИТЕЛЬНОСТИ

Тест, направленный на оценку производительности системы, проводился на примере запуска процесса классификации набора суточных записей со станции BRCR[1]. Проведено 150 запусков. Файлы сигналов (3 канала) не повторялись.

[1] Международные кода сейсмических станций (International Registry of Seismograph Stations (IR), http://www.isc.ac.uk/registries/)

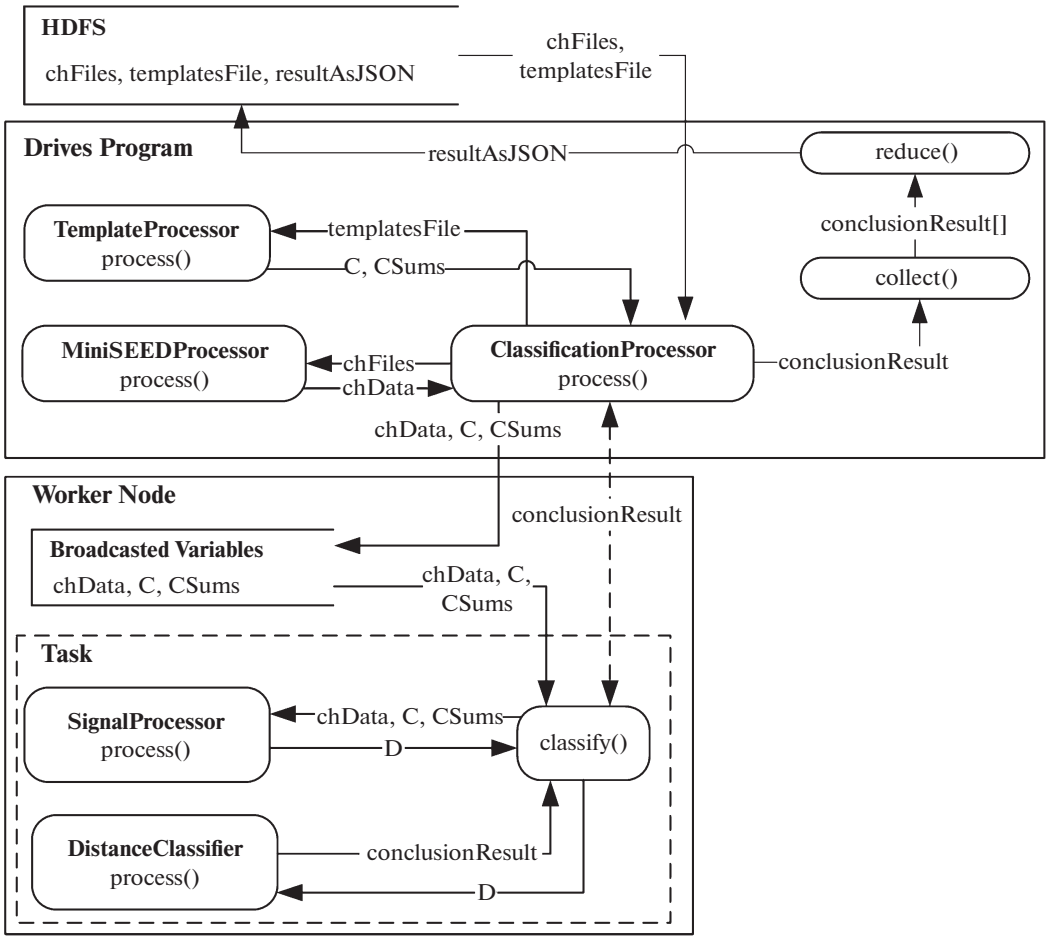


Рис. 9 Диаграмма потоков данных модели (2)–(10) на одном сдвиге расчетного окна в одну секунду.

В таблице 3 указано среднее время работы алгоритма. Представлены 4 программные реализации алгоритма в средах Matlab, Java (консольное приложение) – локальный тест, Java, Python (Spark API приложение) – распределенный тест. Фиксировалось только время расчета от подачи входных параметров (файлы каналов, файл шаблонов, параметры настройки Spark и др.), до получения JSON-файлы карты классификаций.

В качестве пояснения необходимо указать, что сравнение с другими алгоритмами классификаций сейсмических событий, которых насчитывается большое количество, выходит за рамки данной работы и исследования в целом. Не представляется возможным проанализировать математические модели, на основе которых построены эти алгоритмы и их программные реализации. Как следствие, сложно дать оценку возможности запуска в параллельном/распределенном режиме, написать такой программный код и провести его оптимизацию.

Поставленная в рамках рассматриваемого исследования задача решалась с позиции получения оптимального времени исполнения разработанного алгоритма, с целью дальнейшего применения его в режиме потокового приема и обработки сигнала с сохранением точности результатов. Однако даже в сравнение с алгоритмом FAST [16], представленным в обзоре, получено 25-ти кратное увеличение производительности. При этом, увеличение рабочих узлов в кластере и, соответственно, количества ядер, пропорционально будет уменьшаться и время работы алгоритма. Что станет возможным за счет запуска большего числа заданий как на один суточный сигнал, так и увеличения одновременно работающих процессов по количеству сигналов в недельном таймфрейме.

Авторами проведены сравнения с протоколами наблюдений службы геофизического мониторинга Кемеровской области. Полученные результаты в 95% случаев (выборка 2013 года, всего около 500 событий (промышленный взрыв), с двух станций) полностью совпадали по типам заключений.

6. ИСХОДНЫЙ КОД

Исходный код Java-реализации представленного в работе алгоритма, а также исходные данные: шаблоны и файлы суточных сейсмических сигналов, находится в свободном доступе в сети Интернет по адресу https://bitbucket.org/ogidog/seismatica-classifier/src/master/

7. ЗАКЛЮЧЕНИЕ

Разработана программная библиотека для быстрого детерминирования и классификации сейсмических событий в суточном таймфрейме на основе распределенных вычислений в среде Apache Spark. Основываясь на независимости итераций каждого шага сдвига скользящего окна, продемонстрированы подходы к распределению вычислений математической модели алгоритма и ее оптимизации. С использованием специальных транслируемых переменных в общей памяти кластера Spark проводились предварительные расчеты, с целью уменьшения арифметических операций в расчетном задании в схеме организации вычислений Apache Spark.

Проведенные тесты производительности показали существенное уменьшение времени обработки как суточных, так и недельных сейсмозаписей по сравнению с последовательным подходом. Предложенный подход к оптимизации математической модели и программной реализации алгоритма, позволяет применять его для потоковой обработки сигнала ввиду очень малого времени выполнения программного кода.

В работе продемонстрированы способы адаптации предметных математических моделей (сейсмология) к современным технологиям массивно-параллельного исполнения заданий в кластерной инфраструктуре. По мнению авторов, такой подход позволит разрабатывать подобные решения и в других областях научно-технической деятельности.

8. БЛАГОДАРНОСТИ

Исследование выполнено при финансовой поддержке РФФИ в рамках научного проекта № 18-07-00013 А.

СПИСОК ЛИТЕРАТУРЫ

1. *Scarpetta S., Giudicepietro F., Ezin E.C., Petrosino S., Del Pezzo E., Martini M., Marinaro M.* Automatic Classification of Seismic Signals at Mt. Vesuvius Volcano, Italy, Using Neural Networks // Bulletin of the Seismological Society of America. 2005. V. 95. № 1. P. 185–196.
2. *Benbrahim M., Daoudi A., Benjelloun K., Ibenbrahim A.* Discrimination of Seismic Signals Using Artificial Neural Networks // Proceedings of world academy of science, engineering and technology. 2005. V. 4. P. 4–7.
3. *Diersena S., Leeb E.-J., Spearsc D., Chenb P., Wanga L.* Classification of Seismic Windows Using Artificial Neural Networks // Procedia Computer Science. 2011. V. 4. P. 1572–1581.
4. *Hamer R.M., Cunningham J.W.* Cluster analyzing profile data confounded with interrater differences: A comparison of profile association measures. Applied Psychological Measurement. 1981. V. 5. P. 63–72.
5. *Kedrov E.O., Kedrov O.K.* Spectral time method of identification of seismic events at distances of 15°–40° // Izvestiya, Physics of the Solid Earth. 2006. V. 42. № 5. P. 398–415.
6. *Langer H., Falsaperla S., Powell T., Thompson G.* Automatic classification and a-posteriori analysis of seismic event identification at Soufrière Hills volcano, Montserrat // Journal of Volcanology and Geothermal Research. Elsevier. 2006. V. 153. № 1. P. 1–10.
7. *Lyubushin A.A. Jr., Kaláb Z., Častová N.* Application of Wavelet Analysis to the Automatic Classification of Three-Component Seismic Records. Izvestiya, Physics of the Solid Earth. 2004. V. 40. № 7. P. 587–593.
8. *Musil M., Pleginger A.* Discrimination between Local Microearthquakes and Quarry Blasts by Multi-Layer Perceptrons and Kohonen Maps // Bulletin of the Seismological Society of America. 1996. V. 86. № 4. P. 1077–1090.
9. *Ryzhikov G.A., Biryulina M.S., Husebye E.S.* A novel approach to automatic monitoring of regional seismic events // IRIS Newsletter. 1996. V. XV. № 1. P. 12–14.
10. *Shimshoni Y., Intrator N.* Classification of Seismic Signals by Integrating Ensembles of Neural Networks // IEEE transactions on signal processing. 1998. V. 46. № 5. P. 1194–1201.
11. *Ryan T.M., Borisov D., Lefebvre M., Tromp J.* SeisFlows – Flexible waveform inversion software // Computers & Geosciences. 2018. V. 115. P. 88–95.
12. Philippe Lesage. Interactive Matlab software for the analysis of seismic volcanic signals // Computers & Geosciences. 2009. V. 35. № 10. P. 2137–2144.
13. *Jiang W., Yu H., Li L., Huang L.* A Robust Algorithm for Earthquake Detector // Proceedings of the 15 World Conference on Earthquake Engineering. Lisbon. Portugal. 2012.
14. *Álvarez I., García L., Mota S., Cortés G., Benítez C., De La Torre A.* An Automatic P-Phase Picking Algorithm Basedon Adaptive Multiband Processing // IEEE Geoscience and remote sensing letters. 2013. V. 10. № 6. P. 1488–1492.
15. *Madureira G., Ruano A.* A neural network seismic detector // IFAC Proceedings Volumes. 2009. V. 42. № 19. P. 304–309.
16. *Clara E.Y., Ossian O'R., Karianne J.B., Beroza G.C.* Earthquake detection through computationally efficient similarity search // Science Advances. 2015. V. 1. P. E1501057(1–13).
17. *Paul B. Q., Pierre G., Yoann C., Munkhuu U.* Detection and classification of seismic events with progressive multichannel correlation and hidden Markov models // Computers & Geosciences. 2015. V. 83. P. 110–119.

18. IRIS. Incorporated Research Institutions for Seismology. https://www.iris.edu/hq/ (дата обращения: 04.05.2019).
19. *José Emilio Romero*, Manuel Titos, Ángel Bueno, Isaac Álvarez, Luz García, Ángel de la Torre, M Carmen Benítez. APASVO: A free software tool for automatic P-phase picking and event detection in seismic traces // Computers & Geosciences. 2016. V. 90. Part A. P. 213–220.
20. GeoSeisQC. http://www.geoleader.ru/index.php/ru/produkty-ru/geoseicqc (дата обращения: 07.05.2019).
21. ZETLAB Детектор STA/LTA. https://zetlab.com/shop/programmnoe-obespechenie/funktsii-zetlab/analiz-signalov/detektor-sta-lta/ (дата обращения: 07.05.2019).
22. Stratimagic. http://www.pdgm.com/products/stratimagic/ (дата обращения: 07.05.2019).
23. Разработка и создание Грид-приложений для решения прикладных задач геофизики (10-07-00491-а) // РФФИ. http://www.rfbr.ru/rffi/ru/project_search/o_49145 (дата обращения: 07.05.2018).
24. “Использование слабо связанных вычислительных систем для решения обратных задач геофизики (11-05-00988-а) // РФФИ. http://www.rfbr.ru/rffi/ru/project_search/o_43212 (дата обращения: 07.05.2018).
25. Разработка GRID-системы и вычислительных сервисов для исследования геодинамических пространственно-временных процессов по данным ДЗЗ (11-07-12045-офи) // РФФИ. http://www.rfbr.ru/rffi/ru/project_search/o_46676 (дата обращения: 07.05.2018).
26. Distance computations // SciPy.org. https://docs.scipy.org/doc/scipy/reference/spatial.distance.html (дата обращения: 11.05.2018).
27. *Замараев Р.Ю., Попов С.Е., Логов А.Б.* Алгоритм классификации сейсмических событий на основе энтропийного отображения сигналов // Физика Земли. 2016. № 3. С. 31–37.
28. *Замараев Р.Ю., Попов С.Е.* Алгоритм автоматического обнаружения и классификации промышленных взрывов на основе энтропийного отображения сейсмических сигналов // Геофизические исследования. 2019. Т. 20. № 1. С. 38–51.
29. *McKay D.* Information Theory, Inference, and Learning Algorithms // Cambridge: Cambridge University Press, 2003. 631 p.
30. *Kortström J., Uski M., Tiira T.* Automatic classification of seismic events within a regional seismograph network // Computers & Geosciences, 2016. № 87. P. 22–30.
31. *Guojun Gan, Chaoqun Ma, Jianhong Wu*. Data clustering: theory, algorithms, and applications (ASA-SIAM series on statistics and applied probability) // Society for Industrial and Applied Mathematics Philadelphia. PA. USA, 2007. 451 p.

ПРОГРАММИРОВАНИЕ, 2020, № 1, с. 54–62

ТЕОРИЯ ПРОГРАММИРОВАНИЯ: ФОРМАЛЬНЫЕ МОДЕЛИ И СЕМАНТИКА

УДК 004.42

МЕТОДОЛОГИЯ И ИНСТРУМЕНТАЛЬНЫЕ СРЕДСТВА ПРОЕКТИРОВАНИЯ БИНАРНЫХ НЕЙРОННЫХ СЕТЕЙ

© 2020 г. И. В. Степанян[a,b,*]

[a] *Институт машиноведения им. А.А. Благонравова РАН*
101000, Москва, Малый Харитоньевский переулок, д. 4, Россия

[b] *ЗАО "Интеллект"*
115191, Москва, ул. Рощинская 2-я, д. 4, Россия

***E-mail: neurocomp.pro@gmail.com*

Поступила в редакцию 15.02.2019 г.
После доработки 17.04.2019 г.
Принята к публикации 10.06.2019 г.

В статье представлены результаты исследований в области разработки программных методов проектирования, обучения и синтеза бинарных нейронных сетей. В основе исследований лежит модель биоморфного нейрона А.А. Жданова, обладающего помехозащищенностью, возможностью забывания и дообучения. Приведено описание программного инструментария для проектирования и визуализации результатов моделирования бинарных нейронных структур. Приведены примеры и особенности использования формального языка разметки нейросетевых моделей с описанием принципов генерации структур глубокого обучения. Возможности интерпретатора языка разметки нейронных сетей позволяют автоматически генерировать исходный код на языке Verilog с описанием нейроподобной реализации интеллектуальных систем для программных и аппаратных решений на программируемых логических интегральных схемах (ПЛИС).

DOI: 10.31857/S0132347420010069

1. ВВЕДЕНИЕ

В задачах проектирования и алгоритмического синтеза биоподобных нейросетевых структур для интеллектуальной обработки информации важную роль играет синтаксис языка разметки. Синтаксис должен обеспечивать простоту проектирования и наглядность результатов алгоритмической генерации кода и доступность всех необходимых нейросетевых операций синтеза: объединение, мутация, удаление и добавление синаптических связей и пр.

Из нейробиологии известно [1], что входные импульсы нейрона приводят к постепенному увеличению размеров входящих синапсов. При этом растущие синапсы могут подавлять активность соседних синапсов. Таким образом, в структуре нейрона происходит фиксация пространственно-временных комбинаций входных векторов, что лежит в основе механизма обнаружения коррелирующих сигналов. Данный принцип лежит в основе рассматриваемых в данной работе бинарных нейронов и метода автономного адаптивного управления (ААУ) [2], который является одним из подходов к построению адекватных биологическим прототипам моделей нервной системы.

В данной работе описаны нейронные сети на нейронах А.А. Жданова, которые работают исключительно с бинарными данными, передаваемыми на входы и на выходы каждого нейрона и которые не имеют синаптических весовых коэффициентов. Это их кардинально отличает от нейронных сетей, представляющих собой классические нейронные сети (с произвольными значениями на входе и выходе нейронов и весовыми коэффициентами). Особенностью бинарных нейронов А.А. Жданова является то, что для их обучения и настройки не требуется дорогостоящих вычислительных операций умножения, деления и нахождения производной, которые широко применяются в методах обучения классических нейронных сетей. Вместо этого в нейронах срабатывают триггеры-счетчики, которые при достижении определенных условий обеспечивают функционал нейрона, что ускоряет их обучение и приближает к биологическому прототипу.

На базе метода ААУ [2, 3] и описанного в данной работе подхода к его технической реализации, была разработана программно-аппаратная нейроподобная система с возможностью распознавания зашумленных образов [4] и автоматической генерацией нейросетевых структур для ап-

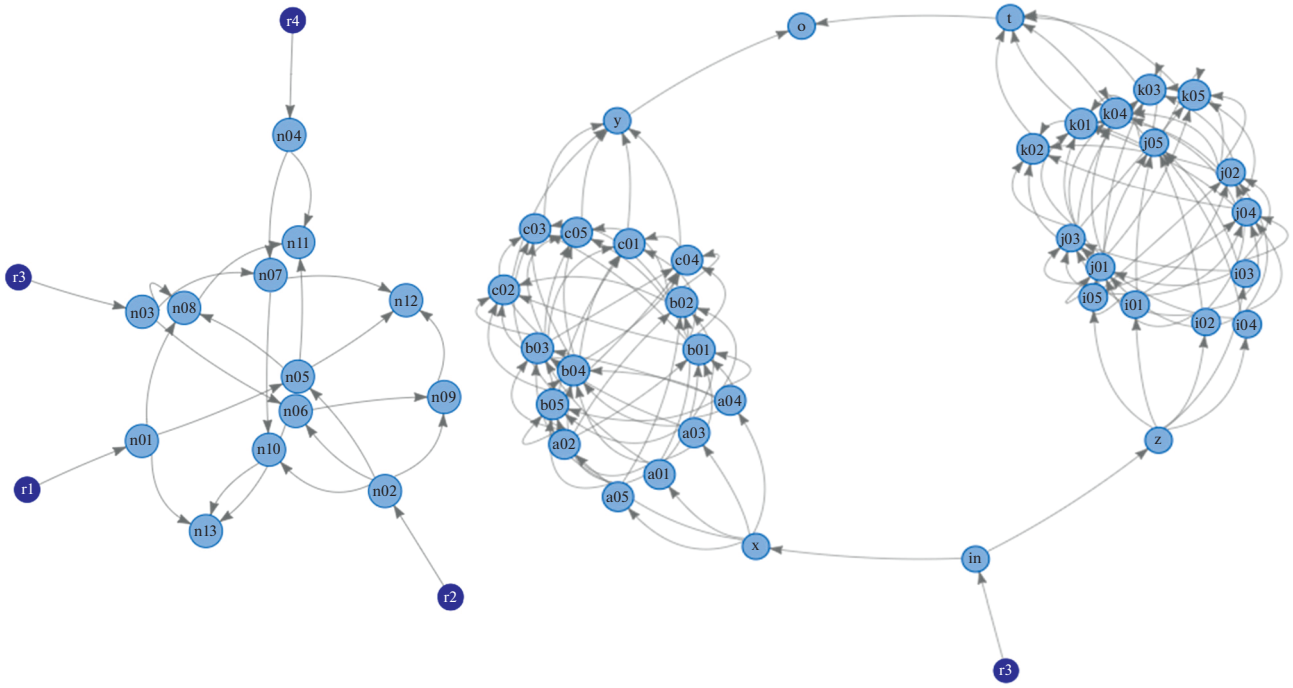


Рис. 1. Пример структуры нейронной сети, состоящей из двух несвязанных между собой подсетей. Затемненные точки – нейроны-рецепторы. Толщина соединительных линий между нейронами зависит от наличия или отсутствия сигнала. Толщина линий от рецепторов не изменяется. Для визуализации использована библиотека d3.js.

парата формирования и распознавания статических образов [5].

2. СТРУКТУРА И ФУНКЦИИ ИНСТРУМЕНТАЛЬНОЙ ПЛАТФОРМЫ ПРОЕКТИРОВАНИЯ БИНАРНЫХ НЕЙРОННЫХ СЕТЕЙ

В целях исследования возможностей алгоритмического синтеза и когнитивных свойств различных структур на бинарных нейронах разработана система автоматизированного проектирования на языке Lua. Эта инструментальная платформа включает в свой состав тройку: логическое ядро (API), интерфейс (GUI) и каталог нейронных сетей.

Основным функциональным блоком разработки является логическое ядро, которое может встраиваться в программные приложения и аппаратные объекты управления как самостоятельный программный модуль. Доступ ко всем функциям ядра осуществляется посредством API. С помощью модулей логического ядра системы разработаны и помещены в каталог нейронных сетей некоторые примеры, что позволяет шаблонизировать разработку нейроподобной реализации интеллектуальных систем. Для упрощения восприятия и администрирования платформы, а также в образовательных целях часть функций логического ядра была привязана к элементам графического интерфейса. Это удобно для отображения нейронных сетей в динамике за счет специально разработанных графических средств визуализации. Административные функции (создание, копирование, редактирование и т.д.) выполняются по умолчанию внутри пользовательского каталога нейронных сетей и могут быть реализованы встроенными средствами администрирования или средствами файловой системы.

Известны языки разметки нейронных сетей для вычислительных нейронаук [6], из которых были заимствованы некоторые идеи. Для нейроподобных алгоритмов был разработан формальный синтаксис, интерпретируемый ядром. В данной работе описан язык разметки биоморфных нейронных сетей, который позволяет описывать нейронные модели на базе бинарных нейронов, хранить и манипулировать ими независимо от конкретной среды моделирования.

3. ОСОБЕННОСТИ ИНИЦИАЛИЗАЦИИ, НЕЙРОНОВ, РЕЦЕПТОРОВ И ЭФФЕКТОРОВ

Бинарные нейроны существенно отличаются от формальных нейронов Маккалока-Питтса и демонстрируют свойства, где обучение и дообучение происходят непосредственно в процессе функционирования сети. Нейрон принимает бинарный вектор и на выходе дает 0, если образ не распознан и 1, если распознан. В целом, модель бинарного нейрона можно сопоставлять с клас-

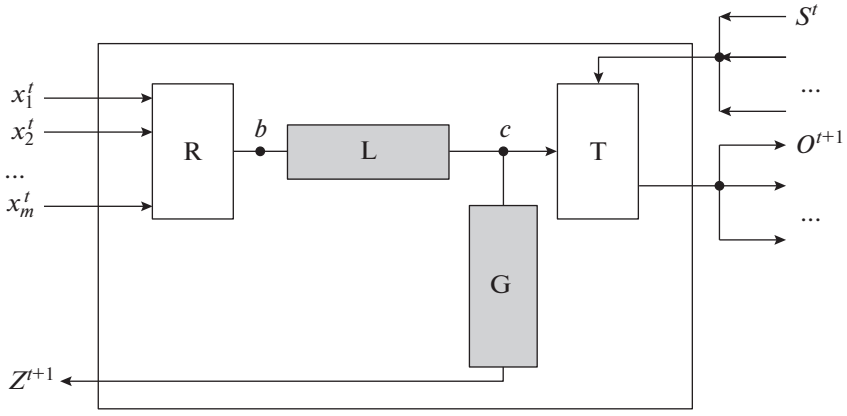


Рис. 2. Структурная схема модели нейрона.

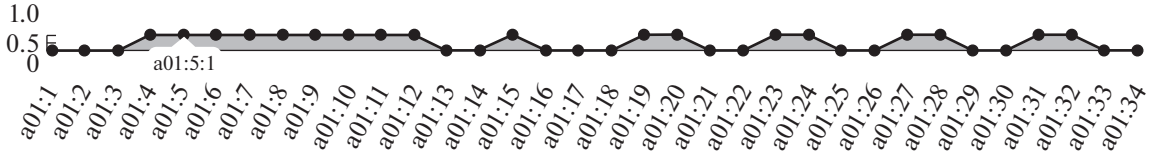


Рис. 3. Пример нейрограммы нейрона a01 с 1 по 34 такты.

сом импульсных (спайковых) нейронных сетей, которые воспроизводят специфичные механизмы нейробиологии [7–9]. Несмотря на то, что теоретические основы спайковых нейронных сетей были предложены еще в 1952 году, они интенсивно развиваются в настоящее время (отметим, что нейросетевые технологии эволюционируют не только в теоретической области нейромоделирования [16, 17], но и в значительной степени в связи с возможностями вычислительной техники[1]).

В соответствии с предлагаемым формальным синтаксисом бинарный нейрон задается своим идентификатором и набором атрибутов. Идентификатор нейрона представляет собой последовательность символов на латинице без пробелов, атрибуты нейрона указываются в скобках через запятую, либо задаются по умолчанию для всех нейронов. Для упрощения читаемости кода и возможности применения различных операций (в том числе автоматических замен) в именах идентификаторов рекомендуют применять префиксы и суффиксы. Это удобно не только для возможности интуитивного понимания нейронной сети, но и для разработки алгоритмов синтеза нейронных структур и алгоритмов путем манипулирования строковыми переменными в различных языках программирования.

При построении сети работает правило: слева от двоеточия – что присоединяем, справа – к чему присоединяем. Рецепторы – это нейроны, которые принимают внешние сигналы. Для передачи входного сигнала любой рецептор может быть подсоединен к любому нейрону из любого слоя сети. Для инициализации рецептора задается его идентификатор и идентификаторы нейронов, к которым подключается рецептор. Описание связей вектора входных значений с нейронной сетью имеет вид:

[1] Пример – взрывной рост нейросетевых технологий распознавания с появлением специализированных графических ускорителей в 2006 г.

```
<ид_входа_1>:<ид_нейрона_11>,<ид_нейрона_12>,…,<ид_нейрона_1j>
<ид_входа_2>:<ид_нейрона_21>,<ид_нейрона_22>,…,<ид_нейрона_2j>,
…,
<ид_входа_n>:<ид_нейрона_i1>,<ид_нейрона_i2>,…,<ид_нейрона_ij>,
```

где `<ид_входа_n>` – идентификатор входного сигнала (рецептора), например, `r01`;

`<ид_нейрона_ij>` – идентификатор нейрона, к которому подключается рецептор, например, `n01`.

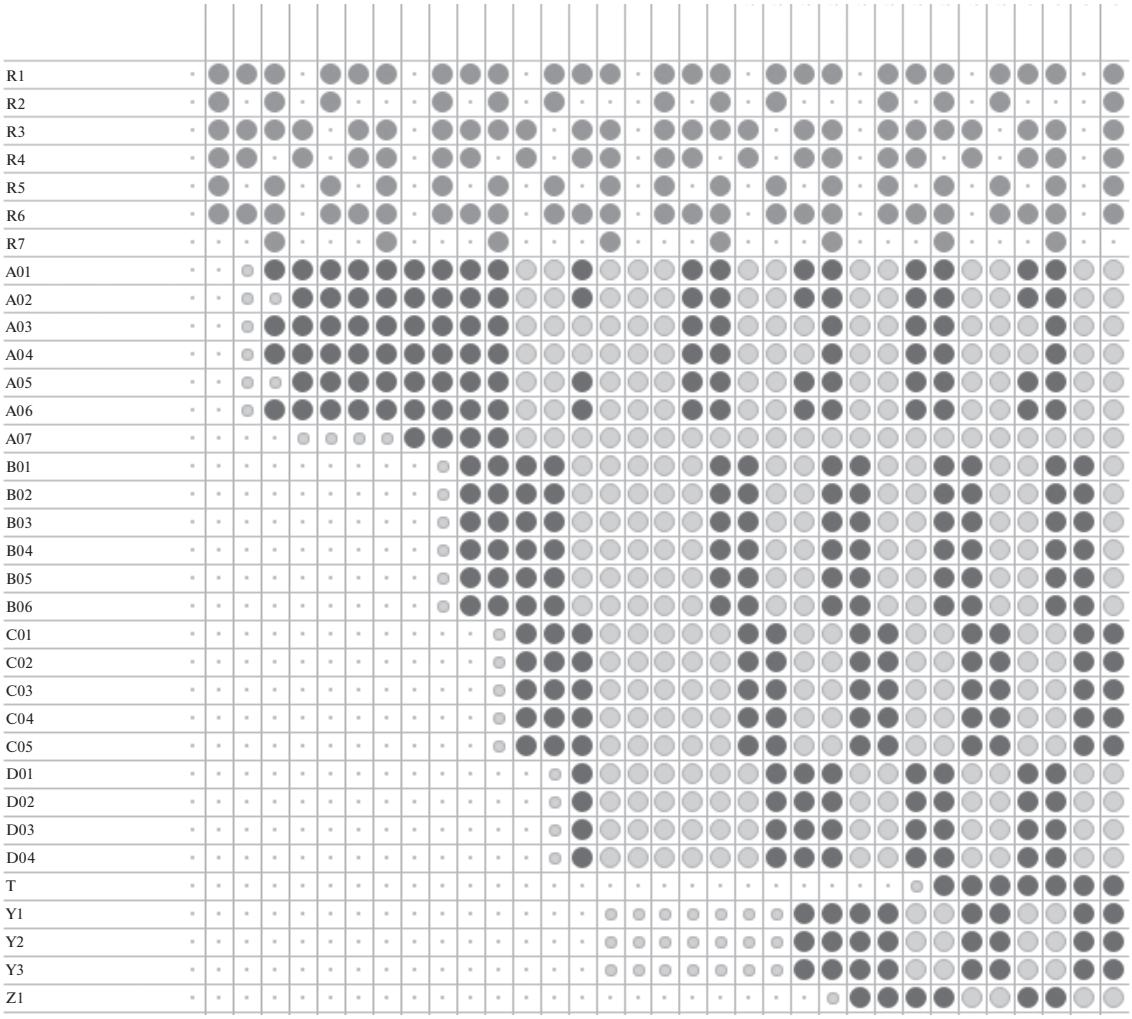


Рис. 4. Фрагмент нейрограммы как результат работы нейронной сети. Идентификаторы рецепторов отображаются по вертикали, номера тактов по горизонтали, слева направо, в прядке возрастания. Цвета и размеры точек соответствуют значениям пороговых счетчиков и активности каждого из нейронов.

Эффекторы – это бинарные нейроны, с которых снимается выходная информация. Эффектор может находиться в любом месте сети. За это отвечает атрибут нейрона “Effector”, принимающий значения “TRUE/FALSE”.

Синтаксис задания матрицы связности, описывающей произвольную структуру нейронной сети идентичен синтаксису инициализации нейронов-рецепторов. Разбиение сети на слои происходит автоматически в результате анализа введенной структуры нейронной сети в соответствии с теорией графов. Пример синтаксиса с фрагментом структуры нейронной сети, где рецептор r01 подключен к трем нейронам:

```
r01:n01,n02,n03
r02:n02
r03:n03
```

В соответствии с введенной структурой программно генерируются графические отображения нейронных сетей. Структура нейронной сети не обязательно должна быть полносвязной. Примеры нейросетевых структур приведены на рис. 1.

4. СТАТИСТИЧЕСКИЕ ПОРОГИ ОБУЧЕНИЯ И ГЕНЕРАЦИИ СИГНАЛА СБРОСА НЕЙРОНА

При достижении заданного статистического порога, отвечающего за число наблюдений образа нейрон активируется и становится обученным. Нейрон будет срабатывать каждый раз, когда на его входы будет поступать соответствующая комбинация входных векторов (все образы кодируются бинарными векторами). Для фиксации состояния обученности нейрона и генерации сброса реализованы статистические пороги L и G соответственно (рис 2).

Таблица 1. Таблица истинности оператора свертки

x1/x2	0	1
0	0	0
1	1	0

На рис. 2 $X^t = \{x_1^t, x_2^t, \ldots, x_m^t\}$ – входной вектор, где – число входов нейрона, – дискретный момент времени. Элемент *R* отрабатывает правило: если нейрон обнаружит вектор, все элементы которого равны 1, то он выдает сигнал 1.

Статистический порог L – это счетчик, который при достижении порогового значения заставляет нейрон срабатывать. Задача этого порога состоит в том, чтобы не пропускать случайные входные вектора. При подаче L раз на этот нейрон вектора, полностью заполненного единицами, нейрон обучится, активируется и начнет передавать единичный сигнал на выход нейрона O при каждой подаче единичного вектора. При распознавании каждого образа, сигнал будет генерироваться до тех пор, пока от нейрона, которому он передает сигнал, не поступит сигнал сброса S.

Триггерный элемент T выполняет функцию кратковременной памяти и удерживает выходной сигнал нейрона до поступления сигнала сброса S. Это необходимо для реализации функции временной задержки в нейронах для обеспечения возможности распознавания динамических явлений и причинно-следственных связей [2].

Статистический порог G – это счетчик, который при достижении своего значения вырабатывает импульс сброса. Его задача – отправить отключающий (тормозящий) сигнал, нейронам, которые передают ему данные. Это необходимо для возможности распознавания неслучайных последовательностей образов.

Структурный порог нейрона L необходим для возможности распознавания зашумленных образов. Например, если нейрон обучен и определенное число раз активировался, то структурный порог в соответствии с заданной функцией упадет, к примеру, до 55%. Это значит, что для активации этого нейрона, больше нет необходимости подавать полный входной вектор и нейрон распознает образ по его части.

Для задания порогов и других параметров нейрона необходимо у идентификатора соответствующего нейрона в скобках задать значения, например: n02(G=4; L=4).

5. НЕЙРОГРАММА – МЕТОД ВИЗУАЛИЗАЦИИ СОБЫТИЙ В НЕЙРОННОЙ СЕТИ

Окно активности нейронов или нейрограмма отображает динамику активации выходных сигналов всех нейронов в процессе обработки образов. Строки с эффекторами на нейрограмме подсвечены. Указатель предельного такта позволяет задать номер такта, до которого будет продолжаться моделирование. Если необходимо обработать все входные данные, то этот указатель должен быть установлен до максимума (по умолчанию). Для моделирования нейронов в сети методом потактной подачи импульсов удобно установить указатель такта на нужный номер и запускать моделирование необходимое количество раз. Это позволит отслеживать динамику информационных процессов (рис. 3, 4) в графическом интерфейсе.

Параметр “такт сброса всех нейронов” позволяет сбросить (разобучить) все нейроны без остановки процесса подачи входных импульсов. Это удобно для отладки нейронных сетей при моделировании на наборах тестовых данных.

Эмоциональная оценка – численный параметр всей нейронный сети, который характеризует общее число активных нейронов на каждом такте моделирования (рис. 4).

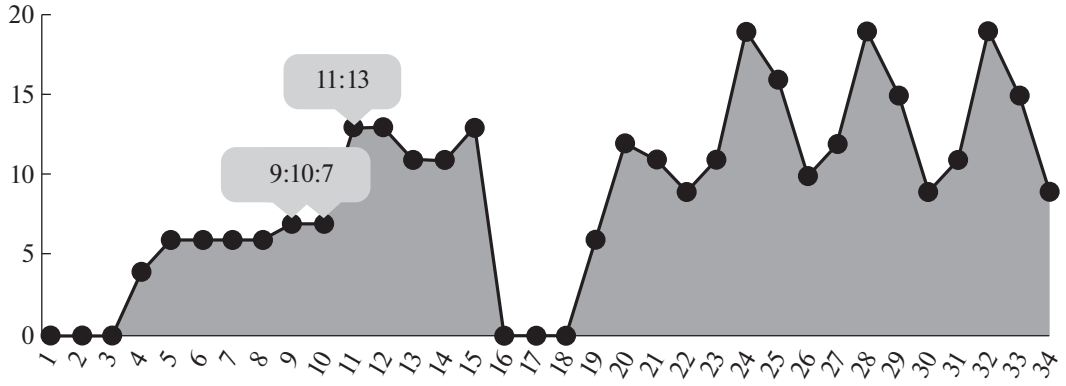


Рис. 5. График эмоциональной оценки нейронной сети. По оси абсцисс номер такта, по оси ординат – число активных нейронов в сети.

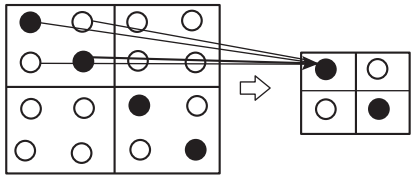


Рис. 6. Пример сжатия признаков во фрагменте слоя нейронов (черная точка – нейрон сработал, белая точка – нейрон не сработал).

Нейросетевые базы знаний согласно [2] являются частью систем ААУ и представляют собой нейронные структуры определенной топологии. Как правило, такие системы оперируют векторами вида „образ-действие-результат“ и могут использоваться для обучаемых систем автономного адаптивного управления в реальной или виртуальной среде совместно с другими блоками (эмоциональной оценки, распознавания и др.). Примеры баз знаний были помещены в каталог нейронных сетей.

Генератор случайных нейронных структур служит для построения случайной структуры нейронной сети без циклов. Данная функция может быть полезна для эволюционно-генетических алгоритмов. Циклы и циклические процессы в бинарных нейронных сетях могут быть полезными, но их роль не изучалась в данной работе.

6. АЛГОРИТМ СИНТЕЗА СТРУКТУР ГЛУБОКОГО ОБУЧЕНИЯ

Была разработана процедура генерации карт признаков, заключающаяся в том, что каждому образу или классу образов ставится в соответствие нейрон со структурой связей, задающей соответствующий образ или класс образов. В отличие от классических нейронных сетей с непрерывными функциями активации данная процедура не итерационная и не рекурсивная, что значительно ускоряет ее выполнение. В задаче идентификации личности, к примеру, такими признаками могут служить глаза и другие характерные элементы лица.

При разработке алгоритма возникла проблема: полученные структуры оказались рассчитанными только на такие виды шумов, которые связаны с отсутствующими элементами образа (слабая помехоустойчивость). Для решения были алгоритмически выделены повторяющиеся фрагменты образов обучающей выборки для формирования и обучения общих промежуточных нейронов. Это не дало повышения качества распознавания зашумленных образов, хотя позволило уменьшить количество нейронов и экономить ресурсы.

Трудности, связанные с помехоустойчивостью нейросетевых структур распознавания привели к идее использования принципов сверточных нейронных сетей, основанных на механизмах распознавания образов в мозге [8, 10]. Исследование структур глубокого обучения позволило разработать подходы к построению нейросетевых структур со свойствами помехоустойчивости и высокой скорости обучения на основе нейронов А.А. Жданова.

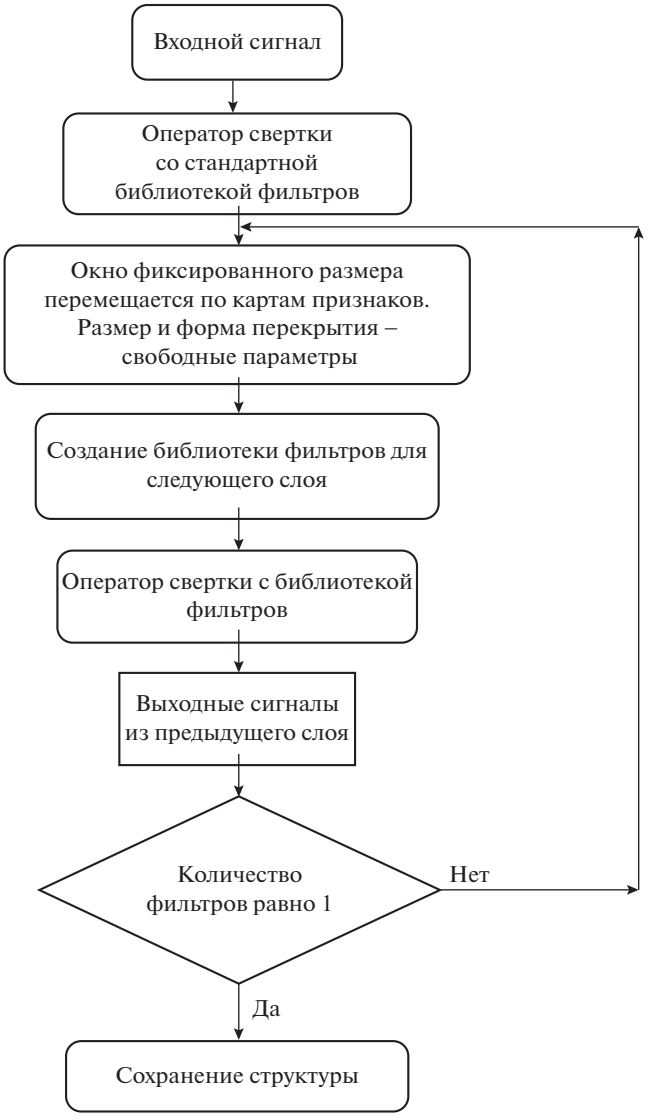


Рис. 7. Рекурсивный алгоритм создания библиотеки фильтров.

Для алгоритма генерации глубокой структуры разработали оператор свертки, который применили при получении значимых признаков для бионических нейронов. Это логическая функция, определяемая таблицей истинности (таблица 1). В таблице 1 × 1 – набор значимых сигналов, ×2 – набор незначимых сигналов.

Поскольку на выходе нейроны имеют бинарный сигнал, то будем считать, что необходимый образ распознан если в рассматриваемом окне (фрагменте образа) распознан хотя бы один об-

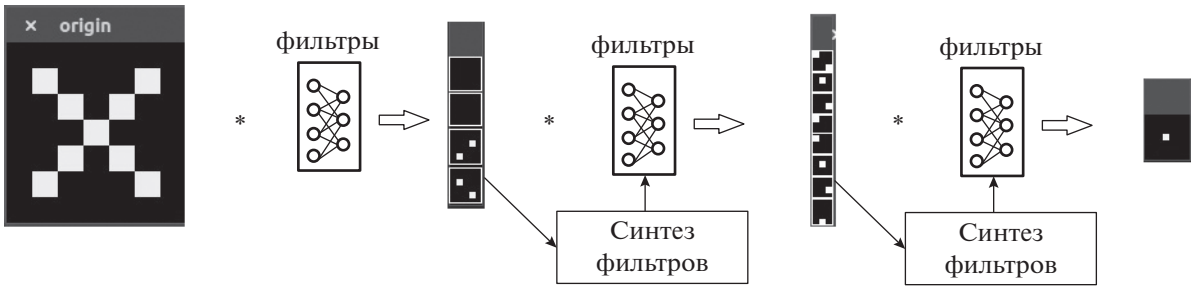


Рис. 8. Схема рекурсивной генерации библиотеки фильтров для тестового образа.

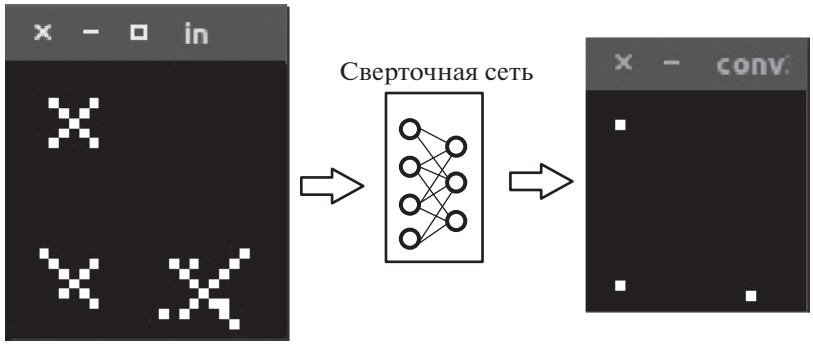


Рис. 9. Примеры распознавания трех зашумленных образов “X” (слева) и идентификация их координат (справа).

раз. При этом размер окна определяется размером нейросетевого фильтра. Далее соединяются все нейроны признаков внутри окна к единому нейрону. Этот нейрон срабатывает, если хотя бы один из входных нейронов сработан. Это условие “слабого” порога или оператора “или”, который реализуется за счет комбинации параметров пороговых функций (рис. 6).

Алгоритм синтеза биоморфной нейроподобной системы распознавания состоит из этапа синтеза структуры подсети библиотеки фильтров для каждого поступающего образа и этапа синтеза сети для распознавания образа. Эти два этапа рекурсивно повторяются, генерируя слой за слоем пока не будет синтезирована нейронная сеть, обеспечивающая за счет своей структуры полное распознавание всего множества заданных образов. Сгенерированная сеть обучается на каждой итерации алгоритма так, что каждый нейрон обучается отдельно. Общая схема алгоритма синтеза структуры подсети библиотеки фильтров-масок для каждого поступающего образа приведена на рис. 7.

Обучение может осуществляться как с учителем и без и использоваться для создания самообучаемой системы распознавания образов. Такая система способна алгоритмически порождать нейронные структуры, обучаемые на наиболее вероятные, по критерию частоты появления образы (задача управления) и на заранее заданные образы (задача распознавания). Алгоритм синтеза сверточной бинарной сети подробно описан в [5], где приведены результаты в области обучения и дообучения сверточных бионических нейронных сетей с применением нейромоделирования.

На рис. 8 схематически изображено применение приведенного алгоритма для создания библиотеки фильтров для распознавания образа “X” и результаты распознавания. Как и в общей задаче распознавания произвольных образов количество фильтров и скрытых слоев заранее неизвестно.

При распознавании в последнем слое были получены условные координаты каждого распознанного образа (рис. 9). Полученные координаты совпадают с геометрическими центрами соответствующих им образов за счет процесса сжатия информации. Зная размер фильтров в каждом слое, можно восстановить исходные образы по условным координатам, что удобно использовать при подсчете распознанных образов на анализируемом изображении.

При тестировании нейроподобных структур использовалось обучающее множество из 60000 рукописных символов из базы данных рукописных цифр MNIST. Доля распознанных символов составила 97.51%. Согласно алгоритму синтеза глубоких структур использовалось разбиение на области с выделением признаков. Хотя современные нейронные сети позволяют достигать менее 1% ошибок, результаты эксперимента показали

работоспособность алгоритма и возможность помехоустойчивого распознавания на бинарных нейронах, то есть без применения операций с плавающей запятой, что существенно. Повысить точность распознавания можно, увеличив разрядность рецепторов. Это повлечет необходимость использования большего количества нейронов. Поэтому в реальных задачах (и особенно в задачах реального времени, где предлагаемые технологии наиболее эффективны) необходим рациональный баланс между быстродействием и точностью.

Подводя итог, отметим, что метод алгоритмической генерации сверточных сетей обладает свойствами, среди которых возможность синтеза сети под выборку, помехоустойчивость к зашумлениям образов, возможность дообучения без необходимости повторного синтеза и повторного обучения, отсутствие сложных вычислительных проблем. Особенностью описанных растущих бионических сетей является возможность семантического анализа (например образы одного класса: “a”, “A”), возможность распознавания трехмерных изображений в различных проекциях, работа с многомерными образами. К недостаткам предложенной методологии можно отнести саму бинарность и как следствие значительное число нейронов при решении реальных задач, что обуславливает необходимость предварительной оценки скоростных параметров алгоритма для управления соотношением время/точность. Предварительная оценка и выбор параметров алгоритма связаны с возможностями специализированного оборудования для функционирования сложных сетей однородных нейронов.

7. VERILOG-КОНВЕРТЕР

Был разработан конвертер, который генерирует Verilog-код нейронной сети исходя из ее формального описания. Описание структуры сети может задаваться вручную, либо генерироваться в автоматическим режиме. Прошивка была успешно проведена на ПЛИС Altera cyclone 4 EP4CE6. Фрагмент генерированного кода:

```
mod_m_counter #(.M(10), .N(4)) baud_gen_unit
(.clk(clk), .reset(reset), .max_tick(tick), .q());
neuron_t1 #(.N(1), .M(2), .Q(2), .PATTERN(1'b1)) n1
(.clk(clk), .reset(reset), .tripping_in(tripping_gl | n5_tripping_out | n8_tripping_out | n13_tripping_out), .s_tick(tick), .din(d [3]),
.tripping_out(n1_tripping_out), .recogn_st(n1_recogn_st), .learn_st(n1learn_st));
neuron_t1 #(.N(1), .M(2), .Q(2), .PATTERN(1'b1)) n2
(.clk(clk), .reset(reset), .tripping_in(tripping_gl | n5_tripping_out | n6_tripping_out | n9_tripping_out | n10_tripping_out), .s_tick(tick), .din(d [2]),
.tripping_out(n2_tripping_out), .recogn_st(n2_recogn_st), .learn_st(n2learn_st));
```

8. ОБСУЖДЕНИЕ И ВЫВОДЫ

В работе приведены подходы к проектированию и описаны элементы программно-аппаратной платформы для разработки, проектирования и прототипирования нейроподобных сетей, описанных в теории “Автономного адаптивного управления” А.А. Жданова. Этот инструментарий обладает следующими функциональными возможностями: проектирование цифровых нейроподобных сетей, анализ информационных процессов в нейроподобных сетях, возможность алгоритмического синтеза нейросетевых структур на бинарных нейронах.

Исследования были проведены на основе разработанного синтаксиса описания нейронных сетей, который позволил создать различные программы, в том числе для синтеза нейросетевых структур и конвертации в Verilog. Исследования продемонстрировали, что предложенный формализм и свойства используемых нейронов позволяют проектировать нейросетевые системы, общим свойством которых является самоорганизация с применением известных в нейроинформатике структур.

Анализируя современные нейросетевые подходы, среди которых сверточные нейронные сети и глубокое обучение [10], можно отметить, что сети на биоморфных бинарных нейронах рассматриваемые в данной статье, могут как существенно отличаться от глубоких нейронных сетей, так и быть с ними связными без теоретических ограничений на комбинации различных нейросетевых решений в нейросетевых ансамблях, включающих самоорганизующиеся, рекуррентные и другие структуры. В частности, для описанного подхода возможно использование динамической структуризации наподобие алгоритмов кортикальных сетей из [11, 12].

От структуры нейронной сети, топологии межнейронных связей зависят качество функциони-

рования, помехоустойчивость и скорость обучения. Динамические свойства бинарных нейроподобных сетей делают их перспективными в системах машинного анализа данных для потоковой обработки информации в реальном времени.

Изложенные материалы носят исследовательский характер и демонстрируют общность подходов к задаче проектирования нейросетевых структур и программных продуктов на их основе. Результаты разработок, демонстрирующих дальнейшее развитие данного направления защищены в патентах [13–15].

СПИСОК ЛИТЕРАТУРЫ

1. *Brewer G.J., Boehler M.D., Pearson R.A., DeMaris A.A., Ide A.N., Wheeler B.C.* Neuron network activity scales exponentially with synapse density, Journal of Neural Engineering. 2008. V. 6. № 1. https://doi.org/10.1088/1741-2560/6/1/014001
2. *Жданов А.А.* Автономный искусственный интеллект – 2 изд-е. М.: БИНОМ. Лаборатория знаний. 2009. 359 с.: ил. (Адаптивные и интеллектуальные системы). ISBN 978-5-94774-995-3
3. *Жданов А.А., Крыжановский М.В., Преображенский Н.Б.* Бионическая интеллектуальная адаптивная система управления мобильным роботом // Искусственный интеллект. 2002. Т. 4. С. 341–350.
4. *Мишустин И.А., Преображенский Н.Б., Жданов А.А., Степанян И.В.* Аппаратная реализация нейроноподобной сети с возможностью распознавания зашумленных образов // Нейрокомпьютеры: разработка, применение. 2018. № 6. С. 19–25.
5. *Степанян И.В., Зиеп Н.Н.* Растущие сверточные нейроподобные структуры для задач распознавания статических образов // Нейрокомпьютеры: разработка, применение. 2018. № 5. С. 4–11.
6. *Vella M., Cannon R.C., Crook S., Davison A.P., Ganapathy G., Robinson H.P., Silver R.A. and Gleeson P.* libNeuroML and PyLEMS: using Python to combine procedural and declarative modeling approaches in computational neuroscience. Frontiers in Neuroinformatics. 2014. V. 8. P. 38.
7. *Ponulak F.* Introduction to spiking neural networks: Information processing, learning and applications. 2010, Acta neurobiologiae experimentalis. V. 71. P. 409–433.
8. *Maas W.* Networks of spiking neurons: The third generation of neural network models. Neural Networks. 1997. V. 10. P. 1659–1671. https://doi.org/10.1016/S0893-6080(97)00011-7
9. *Davies M., Srinivasa N., Lin T.-H., Chinya G., Cao Y., Choday S.H., Dimou G., Joshi P., Imam N., Jain S., Liao Y., Lin C.-K., Lines A., Liu R., Mathaikutty D., McCoy S., Paul A., Tse J., Venkataramanan G., Weng Y.-H., Wild A., Yang Y., Wang H.* Loihi: A Neuromorphic Manycore Processor with On-Chip Learning. IEEE Micro. 2018. V. 38. № 1. P. 82–99. https://doi.org/10.1109/MM.2018.112130359
10. *Schmidhuber J.* Deep learning in neural networks: An overview, Neural networks. 2015. V. 61. P. 85–117.
11. *LeCun Y., Bengio Y.* Convolutional Networks for Images, Speech, and Time-Series, in *Arbib, M. A.* (Eds). The Handbook of Brain Theory and Neural Networks. MIT Press, 1995.
12. *Dileep G., Jeff H.* Towards a Mathematical Theory of Cortical Micro-circuits. October 9. 2009 PLoS Computational Biology, Edited by Karl J. Friston. V. 5. Issue 10. P. E1000532.
13. Свидетельство о государственной регистрации программы для ЭВМ № 2016611299 “Neurox” от 29 января 2016 г. Правообладатель: ЗАО “Интеллект” (RU). Авторы: Степанян И.В., Лекомцев Д.А., Жданов А.А. (Система автоматизированного проектирования биоморфных нейронных сетей Neurox [86450] включена в Реестр по Приказу Минкомсвязи России от 14.12.2016 № 653, Приложение 2, № пп. 12, реестровый № 2393.)
14. Свидетельство о государственной регистрации программы для ЭВМ № 2018661458 от 7.09.2018 “Программа динамического синтеза растущих биоподобных структур на основе искусственных нейронов с расширенной функциональностью и способностью к дообучению для задач распознавания образов и управления динамическими объектами”. Правообладатель: Акционерное общество “Интеллект”(RU). Авторы: Жданов А.А. (RU), Нгуен Нгок Зиеп / Nguyen Ngoc Diep (VN), Перский Г.С. (RU), Пешенко Р.Э. (RU), Пижонков А.Г. (RU), Степанян И.В.(RU), Сямиуллин З.С. (RU).
15. Свидетельство о государственной регистрации программы для ЭВМ № 2018663628 от 01.11.2018 “Программа многоуровневого автономного управления (МААУ) динамическими объектами на основе искусственных нейронов с расширенной функциональностью и способностью к дообучению”. Правообладатель: Акционерное общество “Интеллект” (RU). Авторы: Жданов А.А. (RU), Пешенко Р.Э. (RU), Пижонков А.Г. (RU), Смирнов С.П. (RU), Степанян И.В. (RU).
16. *Савельев А.В.* Нейрокомпьютеры в изобретениях // Нейрокомпьютеры: разработка, применение. 2004. № 2–3. С. 33–49.
17. *Савельев А.В.* Нейросоциометодология проблемы диалога между нейробиологией и нейромоделированием // Нейрокомпьютеры: разработка, применение. 2011. № 1. С. 47–63.

ПРОГРАММИРОВАНИЕ, 2020, № 1, с. 63–74

ЯЗЫКИ, КОМПИЛЯТОРЫ И СИСТЕМЫ ПРОГРАММИРОВАНИЯ

УДК 004.432.4

ПРОМЕЖУТОЧНОЕ ПРЕДСТАВЛЕНИЕ ПРОГРАММ ДЛЯ ОПИСАНИЯ ТИПОВ В ТЕРМИНАХ СОПОСТАВЛЕНИЯ ЗНАЧЕНИЙ С ОБРАЗЦОМ

© 2020 г. В. А. Васенин[a],*, М. А. Кривчиков[a],**

[a] *Московский государственный университет им. М.В. Ломоносова*
119991 Москва, ГСП-1, Ленинские горы, д. 1, Россия
**E-mail: vasenin@msu.ru*
***E-mail: maxim.krivchikov@gmail.com*

Поступила в редакцию 13.05.2019 г.
После доработки 08.06.2019 г.
Принята к публикации 18.06.2019 г.

В статье предлагается промежуточное представление для компактного и обобщенного описания особенностей спецификации системы типов в языках программирования с динамической типизацией, использующее вычисления на уровне типов и построенное на основе сопоставления с образцом. Побудительным мотивом для разработанного авторами промежуточного представления стало промежуточное представление языка Рефал-2 “Язык сборки”. Настоящее промежуточное представление относится не к байт-кодам, а к промежуточным представлениям на основе графа потока исполнения. Такое представление сохраняет информацию о типах значений, что отличает его от языка сборки. Представлена разновидность языка Рефал с поддержкой функций высшего порядка, замыканий и типа данных “ассоциативный массив”, а также транслятор в промежуточное представление. В терминах этого языка приведены примеры описания программ с простыми типами, а также полиморфизма по полям записей. Такие примеры представляют интерес для описания системы типов языков программирования с динамической типизацией.

DOI: 10.31857/S0132347420010070

1. ВВЕДЕНИЕ

В настоящее время предметно-ориентированные языки, как правило, реализуются с использованием динамической проверки типов. В качестве примеров приведем несколько достаточно распространенных предметно-ориентированных языков. Языки командных оболочек операционных систем (Bash) и системы сборки (Make) для описания аргументов и результатов вызываемых ими программ используют подстановку переменных в текстовой форме. Аналогичным образом реализуются языки описания шаблонов, например, Jinja2. Ряд предметно-ориентированных языков для описания конфигурации программных систем (Salt, Ansible) основаны на языках разметки (YAML, JSON) и не содержат средств статической проверки типов. Для некоторых предметно-ориентированных языков, например, языка описания пакетов программ Nix, вопрос о замене динамической проверки типов на статическую ставится уже после их разработки и внедрения в эксплуатацию [1].

Статическую типизацию поддерживают предметно-ориентированные языки следующих трех видов: ограниченные версии существующих языков программирования общего назначения (язык шейдеров GLSL как диалект языка C); встроенные в языки программирования общего назначения с достаточно мощными системами типов и выводом типов (различные eDSL в языке Haskell, например, на основе свободных монад); языки, реализованные с использованием систем макросов языков программирования общего назначения (например, макросы языка Rust).

Основным подходом к реализации проверки типов в языках программирования с динамической типизацией является постепенная типизация [2]. Для более строгой верификации свойств в таких языках системы постепенной типизации зачастую обладают особенностями, обусловленными сложившейся прагматикой (практикой применения) языка. Например, средство mypy, реализующее постепенную типизацию для языка Python, имеет встроенную поддержку так называемых протоколов для структурной типизации [3]. Такие протоколы представляют собой спецификацию, которой должно удовлетворять значение, для его использования в том или ином синтаксси-

ческом контексте (например, в качестве аргумента итерации в конструкции for ... in ... или в качестве аргумента для индексации с использованием квадратных скобок). Язык TypeScript, который интегрирует постепенную типизацию в язык JavaScript, поддерживает ключевое слово keyof в описании типов. Это ключевое слово преобразует объект (ассоциативный массив) в перечисление всех известных ключей, значения которых заданы в этом объекте.

В рамках теории типов наиболее выразительный подход к описанию типов поддерживает исчисление конструкций (Calculus of Constructions), которое позволяет компактно и однородно описать типы с использованием вычислений на уровне типов. В исчислении конструкций типы задаются, фактически, как термы типизированного лямбда-исчисления, т.е. гарантированно завершимые программы. В настоящей статье предлагается промежуточное представление для компактного и обобщенного описания особенностей спецификации системы типов в языках с динамической типизацией, также использующее вычисления на уровне типов. А именно, авторы настоящей статьи предлагают использовать для описания системы типов языков промежуточное представление программ на этих языках на основе сопоставления с образцом. Представленные далее результаты получены в рамках научно-исследовательской работы “Методы и средства разработки верифицируемого программного обеспечения с использованием предметно-ориентированных языков, имеющих заданную формальную семантику”. Вопрос применения этого представления для описания предметно-ориентированных языков более подробно будет рассмотрен в следующих публикациях.

2. ЯЗЫК ДЛЯ РАЗРАБОТКИ СРЕДСТВ ОПИСАНИЯ ПРЕДМЕТНО-ОРИЕНТИРОВАННЫХ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ

На настоящее время наиболее мощные (эффективные) механизмы для сопоставления с образцом в динамически типизированных языках программирования реализованы в языках семейства Рефал [4] (в частности, Рефал-5 [5]). В этой связи в качестве основы для промежуточного представления авторами была разработана разновидность языка Рефал с расширениями. Представленная далее разновидность языка Рефал поддерживает функции высшего порядка и тип данных – неизменяемый ассоциативный массив. Последняя модификация является, насколько известно авторам, новой для языков этого семейства. Синтаксис языка модифицирован с целью более близкого его соответствия языкам программирования, распространенным в настоящее время, что также является новым результатом. Настоящий раздел посвящен описанию особенностей разработанной авторами разновидности языка. Детали процесса разработки и реализации такой разновидности представлены в следующих разделах.

Основным видом постоянных значений в языке являются списки, в которые могут входить элементы перечисленных далее видов, разделенные пробелами.

1. Целочисленные неотрицательные константы (например: 0 12 12345). В настоящей реализации поддерживаются 32-битные целые числа со знаком.

2. Строковые (многострочные) константы в одинарных или двойных кавычках (аналогично языку Python, виды кавычек не различаются) с возможностью экранирования специальных символов (например: “ABC” “A BC\” “ 'A\nB\tC'). Строковые константы интерпретируются как последовательность индивидуальных букв (code-points в терминологии стандарта Unicode; термин “буква” используется в смысле английского слова character для различения с термином “символ”, который используется аналогично английскому symbol).

3. Символы – идентификаторы, которые могут быть ассоциированы с функцией (например: ABC, Fn1). С позиций реализации символы можно охарактеризовать как интернированные строки.

4. Вложенные списки отделяются парами круглых скобок. Например, список “A (B C) D” состоит из трех элементов, первым из которых является символ A, вторым – список из двух символов B и C, и третьим – символ D.

5. Ассоциативные массивы записываются внутри фигурных скобок. Ключами и значениями ассоциативного массива могут быть единичные значения. Ключ отделяется от значения двоеточием, пары “ключ-значение” разделяются запятыми. Например, ассоциативный массив “{abc: DEF, (“12”): (“34”), 12: 34, (1 2): (3 4)}” содержит следующие 4 записи вида “ключ-значение”: символ abc – символ DEF, строка (список букв) “12” – строка “34”, число 12 – число 34, список чисел (1 2) – список чисел (3 4), соответственно. Другие реализации языков семейства Рефал не поддерживают ассоциативные массивы и их сопоставление с образцом.

6. Вложенные функции, объявление которых состоит из ключевого слова fn, опционального имени и тела функции в фигурных скобках.

Отдельным классом выражений в языках семейства Рефал являются образцы. Образцы – это списки, в которых в качестве элементов, в дополнение к отдельным значениям, могут встречаться свободные переменные. Свободные переменные записываются в виде “тип.имя”, как это принято

```
fn Palindrome {
   empty => True;
   s.1 => True;
   s.1 e.2 s.1 => <Palindrome e.2>;
   _ => False;
}
```

Рис. 1. Функция “Паллиндром” на языке из настоящей статьи.

```
Palindrome {
   = True;
   s.1 = True;
   s.1 e.2 s.1 = <Palindrome e.2>;
   e.1 = False;
}
```

Рис. 2. Функция “Палиндром” на языке Рефал-5.

в других языках семейства. В настоящей реализации поддерживаются следующие основные типы свободных переменных:

• “s” – единичное атомарное значение (число, символ, буква);

• атомарное значение заданного типа (“i” – число, “S” – символ, “c” – буква);

• “t” – единичное произвольное значение (атомарное значение или список или ассоциативный массив);

• “d” – ассоциативный массив;

• “e” – произвольная последовательность значений (подсписок).

Образцы для вложенных списков записываются в круглых скобках. Образцы для ассоциативных массивов записываются в фигурных скобках. Для сопоставления ассоциативных массивов с образцом вводятся следующие далее дополнительные синтаксические конструкции на основе оператора “многоточие” (“...”).

• Сохранить остаток ассоциативного массива в новую переменную. Например, следующая конструкция задает образец ассоциативного массива, в котором под ключом “a” должно находиться значение “b”, а остальные пары “ключ-значение”, входящие в ассоциативный массив будут сохранены в новый ассоциативный массив – переменную d.rest: {a: b, ...d.rest}.

• Игнорировать остаток ассоциативного массива. Например, ассоциативный массив {a: b, c: d} не пройдет сопоставление с образцом {a: b}, т.к. содержит дополнительную пару “ключ-значение”, но пройдет сопоставление с образцом {a: b, ...}.

Для удобства чтения программ на языке, пустой список обозначается ключевым словом empty как в значениях, так и в образцах. Кроме того, прочерк (“_”) обозначает безымянную произвольную последовательность значений.

Основным элементом программ на языках Рефал-семейства являются функции. Объявление функции в настоящей реализации начинается с ключевого слова fn, затем следует имя функции, затем в фигурных скобках – последовательность предложений – ветвей сопоставления с образцом, разделенных точкой с запятой (“;”). Предложение начинается с образца, с которым будет сопоставлен аргумент функции, и продолжается с использованием комбинации выражений и связующих операторов, перечисленных далее. Последнее выражение в комбинации определяет результат выполнения предложения. В дополнение к константам и переменным, выражения могут содержать вызовы других функций. Вызов функций, как и в других языках семейства, обозначается угловыми (активационными) скобками, а первый элемент в скобках обозначает вызываемую функцию (например, вызов функции сложения Add для двух чисел 1 и 2 записывается следующим образом: <Add 1 2>).

Классический пример функции “Палиндром” можно представить следующим образом. На рис. 1 показан код на предлагаемом авторами расширении языка Рефал, а на рис. 2 – пример из документации языка Рефал-5 [5].

В этом примере используется связующий оператор “отобразить” (“⇒”), который отделяет выражение слева от результата – выражения справа. В языке поддерживаются представленные далее связующие операторы.

• Операторы безусловного перехода на следующий шаг “запятая” (“,”) или “отобразить” (“⇒”). В других языках семейства, как правило, в качестве такого связующего элемента используется знак равенства. Если при вычислении выражений, следующих за этим оператором, произойдет неуспешное сопоставление с образцом или неуспешный вызов функции, выполнение функции завершится с неуспехом.

• Оператор условного перехода на следующий шаг “амперсанд” (“&”). Если при вычислении выражения, непосредственно следующего за этим оператором, произойдет неуспешное сопоставление с образцом или неуспешный вызов функции, будет рассмотрена следующая альтернатива сопоставления с образцом в рамках активной функции.

• Оператор простого дополнительного сопоставления с образцом “let” (let образец = выражение). Вычисляет значение выражения и сопоставляет его с образцом.

• Оператор дополнительного сопоставления с образцом-функцией “match” (match выражение with функция). Вычисляет значение выражения и

```
fn PreAlph {
  s.1 s.1;
  s.1 s.2 & let e.A s.1 e.B s.2 e.C = <Alphabet>;
}

fn Order {
  (e.1)e.2 & <Pre(e.1)(e.2)> => (e.1)(e.2);
  (e.1)e.2 => (e.2)(e.1)
}
```

Рис. 3. Предикаты на основе успешного/неуспешного завершения вычислений на языке из настоящей статьи.

вызывает функцию с аргументом – значением выражения.

• Новый оператор отрицания “except” (except {предложение}). Если вычисление предложения в исходном контексте значений переменных завершается успешно, результатом оператора отрицания является неуспех. Если же вычисление предложения завершается неуспешно, оператор отрицания возвращает пустой список.

В целом, операторы дополнительного сопоставления с образцом аналогичны условиям и блокам из языка Рефал-5, а разделение перехода на следующий шаг на условный и безусловные аналогично Рефал-6 или Рефал-плюс. Синтаксические отличия настоящей реализации от традиционных обусловлены вопросами эргономики использования языка, в частности, знакомства с языком разработчиков, которые ранее не изучали языки семейства Рефал.

Ограничение распространения сигнала о неуспешном завершении вычислений в рамках одной функции позволяет определять предикаты на значениях в виде функций, вычисление которых завершается успешно для значений, удовлетворяющих предикату, и неуспешно для значений, не удовлетворяющих предикату. В следующем далее примере эту особенность иллюстрирует функция PreAlph, которая проверяет, что первый аргумент (буква) предшествует второму в некотором заданном алфавите. Реализация на языке Рефал-5 возвращает значения-символы T и F, которые обозначают истинное и ложное значение, соответственно. Символы T и F не являются специальными (в отличие, например, от значений True и False в языке Python), их использование обусловлено исключительно соглашением об обозначении логических значений. Реализация в настоящей разновидности языка в качестве истинного значения использует успешное завершение вычислений (возможно, с пустым результатом), а в качестве ложного – неуспешное завершение вычислений, которое в дальнейшем может быть перехвачено с использованием связующего оператора “&”.

```
PreAlph {
  s.1 s.1 = T;
  s.1 s.2, <Alphabet>: e.A s.1 e.B s.2 e.C
          = T;
  e.1 = F;
}

Order {
  (e.1)e.2, <Pre (e.1)(e.2)>:
       { T = (e.1)(e.2):
         F = (e.2)(e.1);
       };
}
```

Рис. 4. Предикаты на основе возвращаемых значений на языке Рефал-5.

Таким образом два примера из главы 4 руководства Рефал-5 [5] в настоящей реализации могут быть записаны следующим образом, с учетом отмеченных выше синтаксических и семантических особенностей (на рис. 3 представлен код на языке, разработанном авторами, а на рис. 4 – код на языке Рефал-5):

Оператор отрицания необходим для адекватного описания нетривиальных условий в системах типов, что необходимо для применения языка по его основному назначению. Пример на рис. 5 показывает применение оператора отрицания для определения предиката “последовательность” из пяти атомарных значений, третье из которых не равно 1”.

Следует также отметить, что в функции, объявленной в операторе match, можно использовать переменные из контекста того предложения внешней функции, в котором был объявлен оператор match. Кроме того, такая функция может быть объявлена с именем, которое может быть использовано внутри нее для рекурсивного вызова или возврата замыкания. Таким образом, настоящая реализация поддерживает функции высшего порядка и замыкания. Пример на рис. 6 демонстрирует применение отмеченных функциональных возможностей языка. Функция Map является функцией высшего порядка, которая применяет пер-

```
ThirdOfFiveNot1 {
  s.1 s.2 s.3 s.4 s.5,except {let 1 = s.3}
}
```

Рис. 5. Оператор отрицания: предикат для последовательности из пяти атомарных значений, третье из которых не равно 1.

вый аргумент-функцию к каждому из следующих аргументов. Функция AddOne определяет переменную s.add, в которой хранится некоторое значение, и безымянную функцию, которая сохраняется в переменной t.addOne и которая добавляет к своему аргументу значение s.add из замыкания. Эта безымянная функция передается как первый аргумент в функцию Map. Результатом выполнения является список, элементы которого увеличены на единицу от исходного (список 1 2 3 4 5 становится списком 2 3 4 5 6).

3. ПРОМЕЖУТОЧНОЕ ПРЕДСТАВЛЕНИЕ ДЛЯ ОПИСАНИЯ ПРЕДМЕТНО-ОРИЕНТИРОВАННЫХ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ

Как правило, для анализа программ используется не их исходный код, а некоторое промежуточное представление с бóльшим, чем у исходного кода, уровнем детализации (уникальность идентификаторов, раскрытие синтаксического сахара), а также с бóльшим уровнем абстракции от оборудования, по сравнению с машинным кодом. В статье [6] приведена следующая далее классификация промежуточных представлений для описания предметно-ориентированных языков программирования.

1. Низкоуровневые промежуточные представления, используемые в компиляторах в машинный код. Например, представления LLVM и Typed Assembly Language.

2. Виртуальные машины уровня приложений на основе байт-кода, ориентированные на императивные языки программирования. Например, машина JVM для языка Java, спецификация CLI для языка C#.

3. Виртуальные машины уровня приложений на основе байт-кода, ориентированные на функциональные языки программирования. Например, машина ZINC для OCaml, машина WAM для Prolog, машина BEAM для Erlang, байт-код языка Python.

4. Внутренние промежуточные представления на основе графа потока исполнения, сохраняющие информацию о типах значений. Например, представление continuation-passing style (CPS).

5. Высокоуровневые, отражающие специфику языка программирования промежуточные представления на основе канонических форм синтаксического дерева. Например, представление STG для Haskell и представление Cminor для компилятора CompCert языка C.

Авторами разработано новое промежуточное представление уровня 4 по представленной классификации с достаточно выразительной для решения задачи системой типов. Побудительным мотивом для настоящего промежуточного представления стало промежуточное представление языка Рефал-2 “Язык сборки” [7].

Основные отличия промежуточного представления в настоящей реализации от языка сборки можно сформулировать в виде следующих тезисов. Язык сборки относится к третьему классу промежуточных представлений, то есть, к виртуальным машинам уровня приложений на основе байт-кода, ориентированным на функциональные языки программирования. Программы на языке сборки – это последовательности команд, не имеющие глубокой внутренней структуры. Предлагаемое промежуточное представление относится к четвертому классу, то есть, к внутренним промежуточным представлениям на основе графа потока исполнения, сохраняющим инфор-

```
fn Map {
   t.fn => emply;
   t.fn s.1 e.rest => <t.fn s.1> <Map t.fn e.rest>;
}

fn AddOne {
   _ =>
     let s.add = 1,
     let t.addOne = fn{s.val => <Add s.add s.val>},
     <Map t.addOne 1 2 3 4 5>
}
```

Рис. 6. Функция высшего порядка и замыкания в языке из настоящей статьи.

мацию о типах значений. Язык сборки описан для языка “Рефал-2”, в предлагаемом промежуточном представлении поддерживаются функции высшего порядка. Как было отмечено выше, сам язык и, как следствие, его промежуточное представление были расширены новыми типами значений и конструкциями для описания системы типов. В качестве достаточно близкого представления можно отметить язык рефал-графов, который используется в суперкомпиляторе SCP-4 [8].

Модель исполнения промежуточного представления имеет следующий вид. При исполнении функции в каждый момент времени задан список значений – аргумент, список значений – результат, последовательность распознанных образцов – значений переменных и набор значений переменных из замыкания, которые индексируются парой целых чисел (порядковый номер внешнего контекста замыкания, номер переменной во внешнем контексте). Контекст исполнения функции содержит также стек обработчиков неуспешных результатов распознавания или вызова функции. Результат может содержать команды вызова (активационные скобки). Перед его сохранением в переменную или перед анализом образцов выполняется стадия активации. На стадии активации все команды вызова в списке заменяются на результаты вызова.

Промежуточное представление хранится в виде дерева в форме s-выражений в текстовом виде (следует отметить, что при необходимости изменить формат хранения, например, на компактную нетекстовую кодировку или XML не составит труда). Дерево содержит узлы-выражения перечисленных далее основных видов.

1. Объявление функции (Function (Name имя функции) выражения ...) и ссылка на функцию (FunctionRef имя функции).

2. Список или структурные скобки: (Structural выражения ...).

3. Вызов или активационные скобки: (Eval выражения ...).

4. Отрицание (Except выражения ...).

5. Ветвление (Branch (выражения ветви 1) (выражения ветви 2) ...).

6. Значение-ассоциативный массив (Dictionary ((выражение-ключ) (выражение-значение)) ...).

7. Значение-константа (Value тип-значения значение).

8. Команды без аргументов: распознавание пустого аргумента (Check), успешное завершение выполнения функции (Return), неуспешное завершение ветки вычислений (Fail), очистка стека обработчиков (PopHandlers), поиск произвольной последовательности значений (Expand), распознавание остатка (Close), активация и сохранение в последовательности переменных текущего результата (Push).

9. Команды распознавания фрагмента аргумента (Recognize направление распознаватель). В качестве направлений поддерживаются: начало и конец аргумента (Left и Right, соответственно); включение в ассоциативный массив в начале аргумента (Dictionary); ключ и значение в ассоциативном массиве (Key и Value номер переменной-ключа, соответственно). Специальные направления (Dictionary и Key) поддерживаются не для всех распознавателей. В качестве распознавателей поддерживаются вложенные списки (Structural), произвольные значения (Term), атомарные значения (Literal), значения заданного примитивного типа (ValueLiteral), константы (Exact), значения ранее распознанных переменных (Old Expression уровень номер).

10. Команда поиска произвольной последовательности значений (Extend).

11. Команда загрузки в аргумент значения переменной для дополнительных условий (Range номер-переменной).

12. Команда вывода в результат постоянных значений (Value тип значение) и значений ранее распознанных переменных (Emit уровень номер).

13. Команды проверки типов (Typecheck IsType имя-функции) и (Typecheck Correct тип-предусловие имя-функции тип-постусловие).

Поле “уровень” в распознавателе OldExpression и комане Emit равно нулю в случае, когда номер соответствует переменной в контексте текущей функции. В противном случае для замыканий оно отражает уровень вложенности лексического контекста, из которого загружается переменная с заданным номером.

Такое промежуточное представление достаточно для трансляции в него программ, написанных на языке, описанном в предыдущем разделе. В качестве одного из направлений дальнейшей работы предполагается добиться большей однородности команд представления. Как уже было отмечено ранее, направления распознавания Dictionary, Key и Value с позиций семантики существенно отличаются от направлений распознавания по списку Left и Right. В частности, с их использованием невозможно эффективное представление перебора пар “ключ-значение”, входящих в ассоциативный массив, для распознавания ассоциативных массивов со сложной структурой. В настоящее время в случаях, когда необходимо решить эту задачу, используются относительно менее эффективные функции стандартной библиотеки, которые выполняют преобразование между ассоциативным массивом и списком пар “ключ-значение”, из которых составлен массив.

Поскольку промежуточное представление предназначено также для описания семантики

программ, написанных на предметно-ориентированных языках программирования, команды промежуточного представления могут содержать дополнительную информацию отладочного характера. Такая информация позволяет установить связь между элементами исходного кода и командами промежуточного представления. Формат дополнительной информации в настоящее время строго не определен, его уточнение является одним из направлений дальнейшей работы.

4. ИНТЕРПРЕТАТОР ПРОМЕЖУТОЧНОГО ПРЕДСТАВЛЕНИЯ

Для исполнения программ, записанных в промежуточном представлении, авторами разработан интерпретатор промежуточного представления. Основными требованиями к интерпретатору являются расширяемость и производительность, достаточная для адекватной поддержки цикла разработки в исследовательском режиме. В этой связи было принято решение о реализации исполнителя языка именно в форме интерпретатора, а не в форме компилятора в машинный код. Интерпретатор реализован на языке Rust, его объем составляет порядка 5 тысяч строк кода.

Интерпретатор использует расширенную (исполнимую) версию промежуточного представления для более эффективной интерпретации. Преобразование хранимой версии промежуточного представления в исполнимую выполняется на стадии предобработки. На этой стадии интерпретатор считывает дерево промежуточного представления из исходного файла и сохраняет его в виде арены – набора линейных последовательностей команд, индексируемых 32-битным беззнаковым целым числом. Таким образом, в памяти такие команды, как Structural или Function хранят индекс последовательности команд в арене. Основная часть процедуры предобработки заключается в выполнении двух следующих процедур. Во-первых, устанавливается соответствие имен функций в командах FunctionRef с индексами их определений в соответствии с лексическим контекстом. Во-вторых, для замыканий выполняется выделение номеров переменных, которые используются замыканием, и команда Function заменяется на специфичную для интерпретатора команду захвата переменных в замыкание CaptureClosure. В перспективе предполагается перенести эту команду в хранимую часть промежуточного представления. Однако на данном этапе отмеченное выше преобразование с технической точки зрения проще выполнять в рамках предобработки в интерпретаторе, а не в трансляторе в промежуточное представление. Кроме того, при предобработке выполняется подгрузка внешних модулей и связывание импортированных функций. Этот аспект реализации носит в большей степени технический характер, его описание выходит за рамки настоящей статьи.

Следует отметить следующие две особенности описываемого в настоящей статье языка, которые нашли свое отражение в промежуточном представлении, и которые оказывают существенное влияние на структуру интерпретатора. Первая особенность связана с процессом исполнения отдельных функций. Семантику команды ветвления Branch в значительной части языков программирования, распространенных в настоящее время, можно достаточно адекватно описать условным оператором if. А именно, если какое-то условие не выполняется, происходит переход к следующей ветви условного оператора. Однако команда поиска произвольной последовательности Extend имеет нетривиальную семантику, которая свойственна в большей степени логическим языкам программирования, и которая обусловлена семантикой переменных типа “e” в языках семейства Рефал. Все случаи неуспешного сопоставления с образцом или вызова функций, которые происходят после выполнения команды Extend возвращают управление обработчику команды Extend, который выполняет “откат” вычислений. Обработчик восстанавливает исходное значение аргумента и переносит следующий элемент аргумента в переменную, после чего возобновляет выполнение операций после команды Extend с новым оставшимся значеним аргумента. Для адекватного описания такого поведения в терминах структурного программирования можно использовать оператор цикла while. Наконец, семантика команды очистки стека обработчиков PopHandlers, которая соответствует связующему оператору безусловного перехода, в совокупности с описанными выше командами Extend и Branch, может быть адекватно описана только в терминах прямых переходов к блокам команд.

Вторая особенность связана с семантикой обработки неуспешных вызовов функций. В классических реализациях языков Рефал-2 и Рефал-5 неуспешный вызов функции приводит к аварийному завершению программы. В настоящей реализации используется принцип, аналогичный обработке программных исключений в языках программирования C++, C#, Java. А именно, неуспешный вызов функции возвращает сигнал неуспешного сопоставления с образцом на уровне вызывающей функции. Как следствие, этот сигнал может быть обработан стандартным образом с помощью обработчиков Branch и Extend. Такую семантику можно сравнить с откатными функциями языка Рефал-Плюс [9]. Разница заключается в той особенности, что решение об обработке или необработке неуспешного вызова функции принимается на уровне вызывающей функции, а не на уровне определения. За счет этой особенности семантика такого аспекта язы-

ка ближе именно к обработке исключений, чем к откатным функциям. В зависимости от последовательности команд вызывающей функции, активация результата может проходить как в конексте вызывающей функции, так и на один уровень выше в стеке вызовов. Последняя ситуация, в частности, имеет место для возвращаемого значения функции, если оно указано после оператора безусловного перехода. Текущая реализация интерпретатора промежуточного представления поддерживает эту особенность в форме частичной хвостовой рекурсии.

Следует также отметить особенности реализации интерпретатора с позиций управления памятью. Интерпретатор представленного в настоящей статье промежуточного представления использует автоматическое управление памятью. Списки значений хранятся с использованием подсчета ссылок. Для повышения эффективности, список значений может храниться как в виде непрерывной последовательности значений (или ссылки на фрагмент другого списка), так и в виде последовательности фрагментов. При добавлении значений распознанных переменных к списку командой Emit используется представленная далее эвристическая схема.

1. Если и последнее имеющееся, и вновь добавляемое значение являются последовательными фрагментами одного и того же списка, вместо пары таких фрагментов записывается один фрагмент, содержащий все необходимые значения.

2. В противном случае, если длина вновь добавляемого фрагмента превышает пороговое значение, фрагмент добавляется в виде ссылки. Размер порогового значения определяется экспериментально из соображений эффективного использования кэш-памяти современных процессоров.

3. Если длина вновь добавляемого фрагмента не превышает пороговое значение, то его элементы копируются в новый список.

На ранних стадиях разработки интерпретатора для автоматического управления памятью использовался алгоритм сборки мусора mark and sweep, причем как для значений, так и для кода промежуточного представления, но его накладные расходы оказались слишком велики. После этого реализация алгоритма сборки мусора была переписана на использование схемы с двумя поколениями. Эта модификация дала ускорение в несколько раз, но производительность интерпретатора все равно была недостаточной при необоснованно большом расходе памяти. В итоге замена сборки мусора на статическую арену для хранения кода и подсчет ссылок для фрагментов значений дала ускорение на 2 порядка, по сравнению с mark and sweep с двумя поколениями, и достаточную, на настоящее время, производительность.

Интепретатор в его настоящей версии выполняет трансляцию транслятора, который описан в следующем разделе, из исходного языка в промежуточное представление за 1.1 секунды с использованием памяти 9.4 Мбайт. Последняя версия транслятора, которая могла выполняться внешней реализацией языка Рефал, которая использовалась в качестве отправной точки для разработки самоподдерживающейся реализации языка, была компилируемой в язык C. Она выполняет аналогичное преобразование за 1.57 секунды с использованием памяти порядка 1 Гбайт.

В перспективе есть возможность дальнейшего повышения эффективности использования кэш-памяти (и, как следствие, повышения производительности интерпретатора) за счет более компактного представления последовательностей значений и команд в памяти. Однако на настоящее время авторы оценивают производительность интерпретатора как адекватную и не считают целесообразным усложнять код интерпретатора с целью дальнейшего повышения его производительности.

Ранее в экспериментальных целях разрабатывались также следующие реализации интерпретатора. Простой рекурсивный интерпретатор деревьев команд оказался неработоспособен из-за отсутствия в распространенных языках программирования поддержки гарантированной хвостовой рекурсии. Поэтому на нетривиальных программах всегда оставалась вероятность столкнуться с переполнением стека. Транслятор в код на языке JavaScript был разработан из тех соображений, что современные реализации языка на основе трассирующих JIT-компиляторов (использовалась реализация V8) обеспечат достаточную производительность при относительно простой схеме трансляции программы. Но на практике такая реализация оказалась неработоспособна из-за особенностей алгоритма сборки мусора в машине V8 и высокого расхода памяти. Интерпретатор на языке Haskell разрабатывался в качестве исполнимой модели денотационной семантики языка. Однако адекватная для применения в дальнейших исследованиях производительность и сравнимый объем кода реализации на языке Rust показали, что в ближайшее время дорабатывать этот вариант нет необходимости.

5. РАЗРАБОТКА ТРАНСЛЯТОРА ИЗ ИСХОДНОГО ЯЗЫКА В ПРОМЕЖУТОЧНОЕ ПРЕДСТАВЛЕНИЕ

Промежуточное представление, показанное в настоящей статье, предназначено, как и любое другое промежуточное представление, для использования автоматизированными средствами, а не для написания программы вручную. Для проверки интерпретатора на работоспособность и для реализации представленной выше разновид-

ности языка авторы разработали транслятор из этого языка в промежуточное представление.

Структура транслятора близка к общепринятой. Исходный код проходит представленные далее по тексту стадии (преобразования).

1. Лексический анализ. Строка из букв разделяется на однородные последовательности – токены (константы, идентификаторы, знаки пунктуации и т.п.).

2. Выделение скобок. В последовательности токенов выделяется структура на основе вложенных пар скобок, соответствующих друг другу.

3. Синтаксический анализ. Структурированное скобками дерево токенов преобразуется в абстрактное синтаксическое дерево исходного языка.

4. Трансляция в промежуточное представление. Абстрактное синтаксическое дерево преобразуется в последовательности команд промежуточного представления. Основная часть трансляции заключается в преобразовании образцов в последовательность команд, аналогично алгоритму трансляции в язык сборки в [7] с некоторыми упрощениями. С учетом используемого способа управления памятью (подсчет ссылок на фрагменты), который позволяет вставлять новые копии фрагментов списков за постоянное время, использовать нетривиальные преобразования для оптимизации построения выходного значения на данной стадии нет необходимости.

5. Свертка ветвей распознавания. Последовательности команд, которые являются общими префиксами для всех ветвей распознавания в коде одной функции, выносятся за пределы команды ветвления.

6. Локальная оптимизация. Относительно простой алгоритм трансляции в промежуточное представление порождает код, в котором можно выделить некоторые шаблоны, допускающие дальнейшее упрощение. Например, в последовательности команд (Close) (Check) последняя команда всегда завершится успешно, поэтому ее можно исключить с сохранением семантики. На стадии локальной оптимизации такие шаблоны в дереве команд приводятся к более простому виду.

7. Сериализация промежуточного представления в выходной текстовый формат. На этом этапе выполняется экранирование строк и вывод промежуточного представления в виде текста с отступами, отражающими структуру кода.

Первая версия транслятора была разработана на языке Рефал-5 и запускалась на одной из реализаций, доступных в сети Интернет в свободном доступе в работоспособном состоянии. Затем транслятор с использованием метода раскрутки (bootstrapping) был переведен в режим самоподдержки (self-hosting) по следующей схеме.

Шаг 1. Транслятор T_0, написанный на языке Рефал-5 скомпилирован компилятором Рефал-5 в исполнимый код E.

Шаг 2. Итеративная доработка интерпретатора и транслятора (T).

a. Получено промежуточное представление и машинный код фрагмента транслятора (E(T') = I', Refal-5(T') = E'). В качестве таких фрагментов использовался код каждой из стадий трансляции, последовательно.

b. Промежуточное представление подается на вход интерпретатору для самоприменения (Interpreter(I')(T')). Результаты его выполнения сравниваются с результатом выполнения машинного кода (E'(T')).

c. При выявлении полученных ошибок и несоответствий производится доработка интерпретатора или транслятора.

d. Итерация завершается после того как достигается самоподдержка:

$E(T) = I$,

$\text{Interpreter}(I, T) = I_1$,

$\text{Interpreter}(I_1, T) = I_2$,

$\text{Interpreter}(I_2, T) = I_3$,

$I_2 \equiv I_3$

Итеративная доработка проводилась в ручном режиме в связи со спецификой задачи. Проверка сохранения самоподдержки по сценарию d добавлена в автоматический набор тестов и выполняется при каждом изменении в интерпретаторе или трансляторе. После достижения самоподдержки, исходная реализация Рефал-5 больше не требуется для запуска модифицированного транслятора, что позволяет перейти к следующему шагу.

Шаг 3. Итеративная модификация синтаксиса и семантики.

a. В интерпретатор добавляется поддержка переходного синтаксиса.

b. Код транслятора T переводится на переходный синтаксис.

c. Удаляется поддержка старого синтаксиса, добавляется поддержка нового синтаксиса.

d. Код транслятора T переводится на новый синтаксис.

На каждом этапе проверяется, что состояние самоподдержки сохранено, путем прямой проверки условия, указанного в пункте d второго шага.

6. ОПИСАНИЕ ПРОСТЫХ ТИПОВ И ПОЛИМОРФИЗМА ПО ПОЛЯМ ЗАПИСЕЙ С ИСПОЛЬЗОВАНИЕМ ПРОМЕЖУТОЧНОГО ПРЕДСТАВЛЕНИЯ

В настоящем разделе приведены примеры применения разработанного промежуточного представления для описания простых типов.

```
fn Bool {
   0;
   1;
}

fn Not {
   0 => 1;
   1 => 0;
}

fn IncompleteNot {
   0 => 1;
}
```

Рис. 7. Код функций над булевскими переменными.

```
(Function (Name Bool) (Branch
   ((Recognize Left Exact Int 0)
      (Check)(Return))
   ((Recognize Left Exact Int 1)
      (Check)(Return))
))

(Function (Name Not) (Branch
   ((Recognize Left Exact Int 0)
      (Check)(Value Int 1)(Return))
   ((Recognize Left Exact Int 1)
      (Check)(Value Int 0)(Return))
))

(Function (Name IncompleteNot) (Branch
   ((Recognize Left Exact Int 1)
      (Check)(Value Int 1)(Return))
))
```

Рис. 8. Промежуточное представление функций над булевскими переменными.

Пример для простых типов представлен на рис. 7, 8. Функция Bool задает спецификацию типа данных, который может принимать одно из двух значений – 0 или 1. Если функцию Not вызывать с аргументом, который удовлетворяет спецификации Bool, то вычисление этой функции обязательно успешно завершится, причем результат вычисления будет удовлетворять спецификации Bool. В свою очередь, функция IncompleteNot определена не для всех возможных значений аргумента, удовлетворяющего спецификации Bool. Для представленных функций эти результаты можно получить автоматически с использованием алгоритма частичного вычисления (прогонки), аналогичного [10]. Следует отметить, что в настоящее время алгоритм прогонки реализован не для всех конструкций промежуточного представления. Использованная для получения результатов настоящей статьи реализация алгоритма прогонки выполняет вычисление значения функции на заданной исходной параметризации аргумента и возвращает два набора параметризаций. Первый (положительный) набор содержит все возможные уточнения исходной параметризации, на которых вычисление функции завершается успешно, вместе с представлением результата вычисления в каждом из уточнений. Второй (отрицательный) набор является дополнением к первому, то есть, описывает все возможные уточнения исходной параметризации, на которых вычисление функции завершается неуспешно.

Полиморфизм по полям записей (row polymorphism) – это свойство системы типов, которое допускает определение функций на ассоциативных массивах, для которых система типов гарантирует невозможность возникновения исключительных ситуаций вида “ключ отсутствует в ассоциативном массиве” и “тип значения не соответствует ожидаемому” [11].

Для определения полиморфизма по полям записей достаточно ввести тип записи и три операции, а именно: проекция (получение значения по ключу, Select), добавление поля (Add), удаление поля (Remove), как показано на рис. 9. Для описания типа записи используется ассоциативный массив. Поведение функции Select описано с использованием аксиоматической семантики в терминах троек Хоара. Предусловие SelectPreCondition и постусловие SelectPostCondition характеризуют поведение функции Select, а именно, тот факт, что для любого (разрешимого) свойства s.predicate, если для некоторого набора значений выполнено условие SelectPreCondition s.predicate, то вычисление функции Select с этим аргументом завершится успешно, а для результата будет выполнено условие SelectPostCondition s.predicate.

В терминах алгоритма прогонки, с учетом его текущих ограничений, процесс проверки этого условия может быть описан в следующей форме. Предположим, что вычисление функции s.predicate всегда завершается с успешным или неуспешным результатом. Выполним прогонку для предусловия с произвольным аргументом: <SelectPreCondition s.predicate e.argument> и оставим только положительный набор параметризаций аргумента. Для всех положительных параметризаций P аргумента e.argument выполним прогонку для исходной функции: <Select P>. Результатом прогонки должны быть только положительные параметризации. В противном случае предусловие задает слишком слабые ограничения, при которых невозможно гарантировать успешное завершение функции Select. Для всех положительных параметризаций Q – результатов

```
fn Select {
   s.label {s.label: t.value, ...} => t.value
}

fn SelectPreCondition  {
   s.predicate s.label { s.label: t.value, ...},
      <s.predicate t.value>
}

fn SelectPostCondition {
   s.predicate t.value, <s.predicate t.value>
}

fn Add {
   {...d.dict} t.value s.label,
      excepr{let {s.label: _,...}=t.dict}
      => {...d.dict,s.label: t.value}
}

fn Remove {
   s.label{s.label: _,...d.rest} => {...d.rest}
}
```

Рис. 9. Реализация полиморфизма по полям записей.

второй прогонки выполним прогонку для постусловия: <SelectPostCondition s.predicate Q>. Результатом этой третьей прогонки, аналогично второй, должны быть только положительные параметризации.

В рамках дальнейшей работы предполагается расширить реализацию алгоритма прогонки для полной поддержки промежуточного представления. Кроме того, предполагается расширить исходный язык двумя конструкциями для проверки типов. Эти конструкции соответствуют инструкциям проверки типов в промежуточном представлении, их предварительный вариант представлен далее.

1. Конструкция “type выражение”, которая проверяет, что выражение является типом, то есть, запускает статическую проверку завершимости выражения для любых входных значений.

2. Конструкция “correct (предусловие) (функция) (постусловие)”, которая для типов “предусловие” и “постусловие” выполняет процедуру проверки, описанную выше.

7. ЗАКЛЮЧЕНИЕ

Одним из важнейших практических аспектов языка программирования является коммуникация между разработчиками. Исходный код содержит наиболее точную и актуальную информацию о внутреннем устройстве системы. Для новых языков программирования элемент схожести их синтаксиса с другими языками, с которыми знаком разработчик, способствует более быстрому пониманию кода на этом языке [12]. Кроме того, как следует из результатов [12], язык без ключевых слов или с активным использованием вместо них неочевидных спецсимволов представляется менее интуитивным, чем язык с ключевыми словами. Личный опыт работы авторов со студентами механико-математического факультета МГУ согласуется с этими результатами.

В части известных авторам аналогов предлагаемого подхода к проверке типов следует отметить работу В.И. Шелехова по предикатному программированию [13]. Языки семейства Рефал и суперкомпиляция использовались ранее для верификации свойств, которые имеют характер низкого уровня абстракции от аппаратного обеспечения [14, 15]. В статье [16] приводится пример использования суперкомпиляции в рамках теории типов Мартина-Лёфа. В общем контексте исследования, результаты которого приведены в настоящей статье, следует упомянуть язык исполняемых программных спецификаций [17], который предназначен для спецификации предметно-ориентированных языков. Описание типов с использованием функций используется также в системе PVS (Prototype Verification System) [18], которая используется в упомянутой выше статье [13].

В качестве направления дальнейшей трансформации синтаксиса с целью повышения его эргономичности авторы рассматривают возможность ис-

пользования более привычных обозначений для списков и переменных-образцов. В значительной части распространенных языков программирования литералы списков обозначаются квадратными скобками. Исключение составляют языки семейства LISP, которые можно причислить к относительно нераспространенным. Использование квадратных скобок для списков освободит круглые скобки для их общепринятой роли – конструкции группировки. Для обозначения переменных предполагается использовать только их имя с опциональной аннотацией типа в скобках. Переменные, обозначающие образцы распознавания произвольной последовательности, в этом случае будут обозначаться префиксом-многоточием (таким образом, например, образец (s.a e.b s.a) будет записан как [a ...b a] или [(a: atom) ...b a]). Вопрос записи, допускающей различие между символами и переменными-образцами в настоящее время требует дополнительной проработки. Одно из возможных решений этого вопроса заключается в использовании подхода на основе регистра первой буквы идентификатора, аналогичного используемому в языке Haskell: в переменных первая буква должна быть строчной, а в символах – заглавной.

8. БЛАГОДАРНОСТИ

Работа выполнена в рамках проекта РФФИ 16-07-01178а.

СПИСОК ЛИТЕРАТУРЫ

1. *Hufschmidt T.* A type-system for Nix [Электронный ресурс] // NixCon 2017. 2017. URL: http://nixcon2017.org/schedule.nixcon2017.org/system/event_attachments/attachments/000/0000/003/original/main.pdf
2. *Siek J., Taha W.* Gradual Typing for Objects // ECOOP 2007 – Object-Oriented Programming / под ред. Ernst E. Springer Berlin Heidelberg, 2007. P. 2–27.
3. *Levkivskyi I., Lehtosalo J., Langa Ł.* PEP 544 – Protocols: Structural subtyping (static duck typing) [Электронный ресурс] // Python.org. 2017. URL: https://www.python.org/dev/peps/pep-0544/(дата обращения: 10.01.2019).
4. *Турчин В.Ф.* Метаалгоритмический язык // Кибернетика. 1968. № 4. С. 45–54.
5. *Turchin V.F.* REFAL-5, Programming Guide and Reference Manual. Holoyke, MA: New England Publishing Co., 1989.
6. *Васенин В.А., Кривчиков М.А.* Методы промежуточного представления программ // Программная инженерия. 2017. Т. 8. № 8. С. 345–353.
7. *Романенко С.А.* Машинно независимый компилятор с языка рекурсивных функций: дисс. ... канд. физ.-мат. наук: 01.01.10. Москва: ИПМ АН СССР, 1978. 148 с.
8. *Немытых А.П.* Суперкомпилятор SCP-4: общая структура. М.: Изд-во ЛКИ, 2007. 150 с.
9. *Романенко С.А., Гурин Р.Ф.* Язык программирования Рефал Плюс. Переславль-Залесский: Ун-т города Переславля им. А.К. Айламазяна, 2006. 221 с.
10. *Турчин В.Ф.* Эквивалентные преобразования рекурсивных функций, описанных на языке РЕФАЛ. Киев-Алушта. 1972. С. 31–42.
11. *Wand M.* Type inference for record concatenation and multiple inheritance // Information and Computation, 1991. V. 93. № 1. P. 1–15.
12. *Stefik A., Siebert S.* An Empirical Investigation into Programming Language Syntax // Transactions on Computing Education, 2013. V. 13. № 4. P. 19:1–19:40.
13. *Шелехов В.И.* Верификация и синтез эффективных программ стандартных функций в технологии предикатного программирования // Программная Инженерия. 2011. № 2. С. 14–21.
14. *Немытых А.П.* Верификация как параметризованное тестирование (эксперименты с суперкомпилятором SCP4) // Программирование. 2007. Т. 33. № 1. С. 22–35.
15. *Лисица А.П., Немытых А.П.* Об одном приложении вычислений с оракулом // Программирование. 2010. Т. 36. № 3. С. 43–53.
16. *Ключников И.Г., Романенко С.А.* Суперкомпиляция для теории типов Мартина-Лёфа // Программирование. 2015. № 3. С. 73–87.
17. *Новиков Ф.А., Новосельцев В.Б.* Язык исполняемых программных спецификаций // Программирование. 2010. Т. 36. № 1. С. 66–78.
18. *Shankar N., Owre S.* Principles and Pragmatics of Subtyping in PVS // Recent Trends in Algebraic Development Techniques, WADT'99 / под ред. Bert D., Choppy C., Mosses P.D. Toulouse, France: Springer-Verlag, 1999. V. 1827. P. 37–52.