УПРАВЛЕНИЕ, ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА И ИНФОРМАТИКА

УДК: 51-74

ПОВЫШЕНИЕ ЭФФЕКТИВНОСТИ ВЕКТОРИЗАЦИИ КОНСТРУКТОРСКОЙ ДОКУМЕНТАЦИИ ЭЛЕКТРОННЫХ СРЕДСТВ

ТЕМНОВ Кирилл Анатольевич, старший преподаватель Московского авиационного института (государственного технического университета).
Тел.: +7-916-962-91-32, e-mail: allselead@gmail.com

TEMNOV Kirill A. Moscow Aviation Institute (State Technical University) Lecturer.
Tel.: +7-916-962-91-32, e-mail: allselead@gmail.com

Современные векторизаторы имеют невысокую эффективность распознавания конструкторской документации электронных средств. В статье предлагается алгоритм, повышающий эффективность векторизации за счет правильной маркировки текстовых и графических компонентов бинарных изображений конструкторской документации. Предлагаемый алгоритм позволяет получить выигрыш по совокупности своих свойств, по сравнению с последовательным применением алгоритмов маркировки компонентов изображения и алгоритмов выделения текста на изображении. Рассматриваемый алгоритм позволяет с большой надежностью выделять компоненты текста в зашумленных изображениях.

Modern vectorizers have low efficiency of recognition of the design documentation of electronic. In this paper I propose an algorithm that increases the efficiency of tracing through the proper labeling of text and graphic components of the binary images of the design documentation. Application of quadtrees for labeling and extracting text and graphics components on binary images with the following clustering of labeled components to text and graphics is described in the article. Proposed algorithm gains an advantage over sequential application of image components labeling algorithms and algorithms of text extracting in the image. Described algorithm allows to select the text components in noisy images with high reliability.

Ключевые слова: векторизация, конструкторская документация электронных средств, квадрантные деревья, выделение текстовых компонентов, маркировка бинарных изображений.

Key words: vectorization, design documentation for electronic, quadtrees, text components extracting, binary images labeling

Введение

Маркировка компонентов является важным этапом в распознавании изображений и применяется совместно с алгоритмами векторизации или оптического распознавания текста. Особенностью предлагаемого алгоритма является возможность быстрой кластеризации маркированных областей на текстовые и графические.

Использование квадрантных деревьев позволяет увеличить производительность алгоритмов, связанных с обработкой бинарного изображения. Квадрантные деревья как структура данных, а также

операции с ними подробно описаны в [1]. Маркировка связных компонентов с помощью квадрантных деревьев рассмотрена в [2] и [3].

Предлагаемый метод использует квадрантные деревья для хранения данных об изображениях и операций над этими данными. Квадрантное дерево, содержащее информацию об изображении, представляет собой структуру данных, которая может быть теоретически маркирована быстрее, чем исходное изображение, представляющее собой линейную структуру.

Предлагаемый алгоритм разработан специально для векторизации бинарных изображений чертежей, содержащих достаточное количество графической и текстовой информации. В следующих разделах описываются части предлагаемого алгоритма: построение квадрантного дерева, алгоритм маркировки и алгоритм кластеризации.

Описание частей алгоритма производится с помощью процедур, реализованных на псевдокоде.

Построение квадрантного дерева

Квадрантные деревья, как структура данных, были предложены R. Finkel и J.L. Bentley в 1974 г. Квадрантные деревья по структуре похожи на бинарные деревья поиска, за исключением того, что каждый узел может иметь до четырех подузлов.

Любое бинарное изображения размером $m \times n$ может быть представлено в виде полного квадрантного дерева с максимальным количеством листовых вершин $2^{\left[\log(\max(m,n)+1)\right]}$. Высота такого дерева будет равна*

$$H = \left[\log\left(\max\left(m,n\right)\right)\right] + 1. \qquad (1)$$

Полное квадрантное дерево имеет максимально возможное количество вершин для квадрантного дерева с заданной высотой. Полные квадрантные деревья используются для оценки затрат памяти и скорости работы алгоритмов сверху (по максимальным значениям), а при реализации алгоритмов применяются неполные квадрантные деревья (рис. 1).

Для описания работы алгоритма будем считать, что для построения квадрантного дерева используется изображение с белым цветом фона и черным цветом переднего плана. Длина и ширина изображения часто не совпадают с размером основания соответствующего квадрантного дерева. Оставшееся пространство, дополняющее размеры изображения до необходимых, не используется при вычислениях, поэтому можно полагать, что оно заполнено фоновым (белым) цветом.

Процедура отображения бинарного изображения на квадрантное дерево заключается в создании листов дерева, соответствующих размеру одного пиксела изображения, на нижнем уровне дерева. Листы более высоких уровней окрашиваются в цвет своих потомков. Если потомки имеют и черный, и белый цвет, их родительские листы считаются серыми. Цвет каждого серого узла дерева вычисляется как сумма цветов дочерних узлов, а узлы, имеющие белый цвет, считаются листовыми и не имеют дочерних подузлов.

Построение квадрантного дерева начинается с корня. Корнем принято считать элемент дерева, не имеющий родительского элемента. Будем считать, что корень дерева имеет уровень 1, и все последующие уровни нумеруются через 1.

Для построения квадрантного дерева необходимо занести в него информацию о расположении пикселов переднего плана. Таким образом, чтобы заполнить дерево, необходимо добавить в него соответствующие узлы. Процедура ADD_NODE описывает процесс добавления нового узла в дерево. Процедура CREATE_CHILDS, вызываемая из ADD_NODE всегда создает четыре дочерних узла

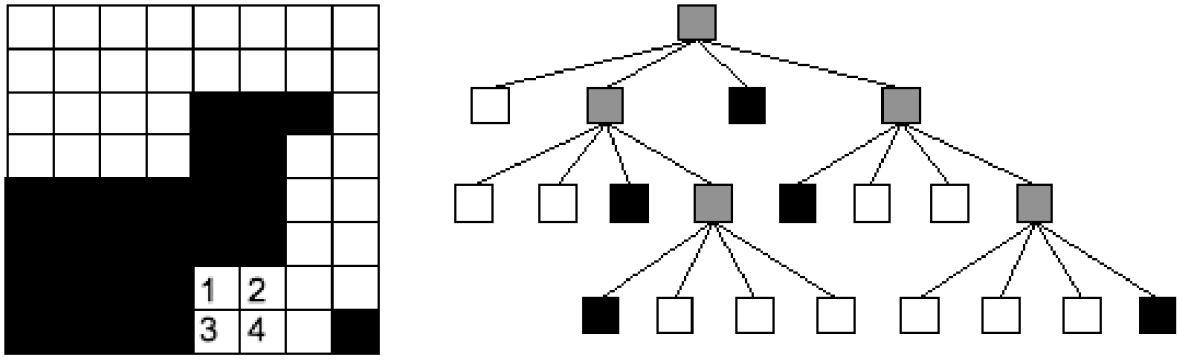

Рис. 1. Отображение бинарного изображения на квадрантное дерево

* Квадратные скобки обозначают взятие ближайшего к нулю целого числа.

квадрантного дерева. Создание четырех дочерних узлов ведет к дополнительному расходу памяти, но позволяет сократить время работы предлагаемого алгоритма за счет оптимизации алгоритма обхода дерева.

Обход дочерних узлов дерева в процедурах осуществляется в порядке, представленном на рис. 2. Цифрами обозначены номера дочерних узлов. При изменении порядка обхода должны быть изменены и все процедуры, осуществляющие обход дерева, но сложность алгоритмов при этом никак не изменится.

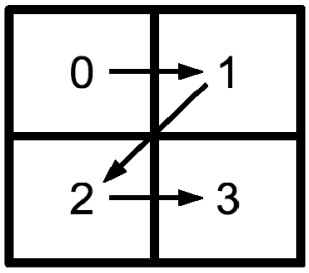

Рис. 2. Направление перемещения по дочерним узлам квадрантного дерева при обходе

После построения дерева вычисляется количество пикселов переднего плана, входящих в каждую вершину (цвет вершины). Процедура RECALC_COLORS производит рекурсивный обход дерева, начиная с корня, и вычисляет цвет каждой вершины дерева.

Алгоритмы построения квадрантного дерева имеют сложность $O(n)$ [4—6].

```
procedure ADD_NODE(ROOT, OFFSET_X, OFFSET_Y):
    # Процедура добавления нового пиксела (и соответствующей ему
вершины)
    # в квадрантное дерево
    begin
        if ROOT.SIZE == 1 then
        begin
            ROOT.COLOR = BLACK
            return
        end
        if not HAVE_CHILDS(ROOT) then
        begin
            ROOT.CHILDS = CREATE_CHILDS(ROOT)
        end
        if 0 <= OFFSET_X < (ROOT.SIZE / 2) and
                0 <= OFFSET_Y < (ROOT.SIZE / 2) then
        begin
            ind = 0
        end
        else if (ROOT.SIZE / 2) <= OFFSET_X < ROOT.SIZE and
                0 <= OFFSET_Y < (ROOT.SIZE / 2) then
        begin
            ind = 1
            OFFSET_X = OFFSET_X - (ROOT.SIZE / 2)
        end
        else if 0 <= OFFSET_X < (ROOT.SIZE / 2) and
            (ROOT.SIZE / 2) <= OFFSET_Y < ROOT.SIZE then
        begin
            ind = 2
            OFFSET_Y = OFFSET_Y - (ROOT.SIZE / 2)
        end
        else if (ROOT.SIZE / 2) <= OFFSET_X < ROOT.SIZE and
                (ROOT.SIZE / 2) <= OFFSET_Y < ROOT.SIZE then
        begin
            ind = 3
            OFFSET_X = OFFSET_X - (ROOT.SIZE / 2)
            OFFSET_Y = OFFSET_Y - (ROOT.SIZE / 2)
        end
        ADD_NODE(ROOT.CHILDS[ind], OFFSET_X, OFFSET_Y)
    end
```

```
procedure RECALC_COLORS(NODE):
    # Процедура вычисления цвета узла дерева.
    # Цвет узла дерева равен цвету всех дочерних узлов данного узла.
    begin
        if not HAVE_CHILDS(ROOT) then
        begin
            if not ROOT.SIZE == 1 then
            begin
                ROOT.COLOR = 0 # WHITE
            end
        end
        else
        begin
            col = 0
            foreach node in ROOT.CHILDS do
            begin
                col = col + RECALC_COLORS(node)
            end
            ROOT.COLOR = col
        end
        return ROOT.COLOR
    end
```

Алгоритм маркировки связных компонентов квадрантного дерева

Для операций над узлами квадрантного дерева применяется алгоритм поэлементного рекурсивного обхода. Ввиду того что глубина квадрантного дерева, представляющего бинарное изображение, является небольшой величиной*, обход в глубину является более быстрой альтернативой поэлементного обхода каждого уровня дерева.

Описываемый алгоритм маркировки — четырехсвязный, т.е. при обходе каждого узла производится поиск значений цвета четырех соседних узлов, не являющихся дочерними для какого-либо одного узла дерева. Алгоритм предназначен специально для обработки чертежей, а ввиду того что толщина линий на бинарных изображениях чертежей больше 4 пикселов, четырехсвязности достаточно для маркировки связных компонентов. Алгоритм может быть модифицирован в восьмизвязный, однако это приведет только к дополнительной потере скорости, и восьмизвязный алгоритм в работе не рассматривается.

Исходя из оптического разрешения выбирается размер узла дерева, являющегося базовым для алгоритма маркировки. Назовем этот узел блоком.

В зависимости от оптического разрешения изображения размер блока (его стороны) может варьироваться от 2 до 16 пикселов. Чем меньше размер блока, тем точнее маркированная область будет воспроизводить форму объектов и тем больше вычислений потребуется для алгоритма маркировки. При маркировке изображений в небольшом оптическом разрешении (до 150 dpi) предпочтительным является размер блока, равный двум, так как велика вероятность того, что в блок большего размера попадет несколько областей изображения, которые впоследствии будут маркированы как одна.

Алгоритм маркировки последовательно проходит все блоки дерева, отличные от белого**. Для каждого узла проверяются наличие соседних узлов и их цвет. Если соседние блоки также имеют серый цвет, им присваивается текущая метка. Наиболее сложным в алгоритме маркировки является получение пути соседних узлов.

Алгоритм получения пути соседних узлов

Алгоритм использует информацию о пути к текущему*** узлу дерева. Путь представляет собой список, состоящий из индексов узлов, начиная с корня дерева, которые необходимо пройти, чтобы достигнуть текущий узел. На рис. 3 путь к темно-серому узлу дерева будет представлять список из вершин [2, 1, 1] предыдущих уровней, начиная со второго. Путь однозначно определяет положение узла в дереве.

Полурекурсивный алгоритм маркировки работает с четырехсвязными компонентами, поэтому

* Квадрантное дерево для изображения размером 10000 × 10000 пикселов имеет высоту 14 уровней.

** Для больших блоков необходимо задавать большие значения цвета.

*** Для ясности будем называть узел дерева, для которого необходимо получить соседние узлы, текущим.

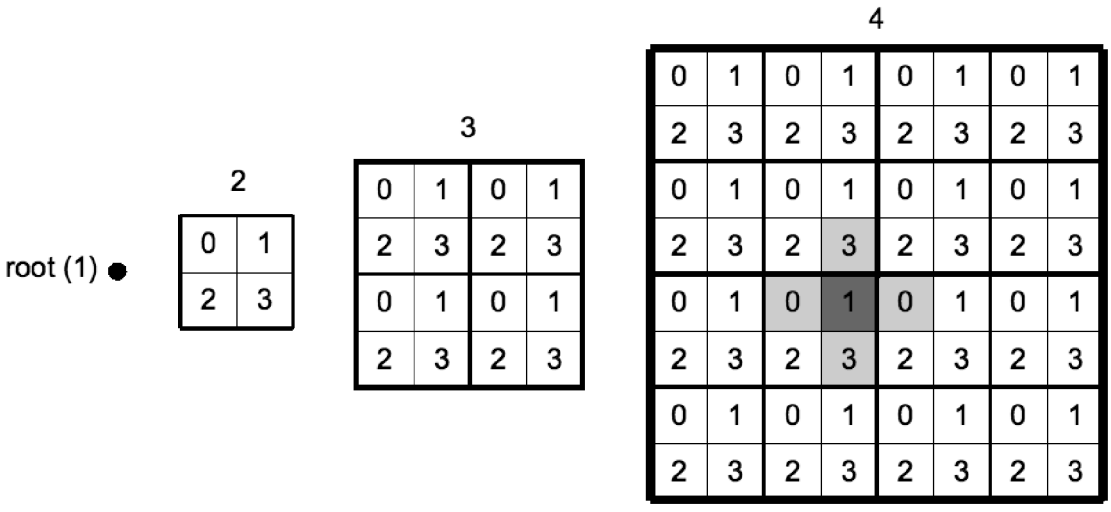

Рис. 3. Получение соседних узлов в квадрантном дереве

необходимо получить путь к соседним узлам, которые находятся сверху, снизу, слева и справа от текущего узла, если смотреть на срез уровня, в котором находится узел. На рис. 3 соседними узлами для темно-серого узла на 4 уровне являются 4 светло-серых узла.

Алгоритм получения пути соседних узлов имеет сложность $O(n)$, где n — глубина квадрантного дерева.

Ввиду того что родительский узел текущего узла всегда имеет 4 дочерних, получение путей двух узлов производится заменой последнего значения в пути на номера этих узлов. Для примера на рис. 3 получим пути [2, 1, 0] и [2, 1,3]. Оставшиеся два пути получаются по алгоритму, описанному процедурой GET_ADJACENT_NODE_PATHES.

Алгоритмы TOP_NEIBHOUR, LEFT_NEIBHOUR, RIGHT_NEIBHOUR и BOTTOM_NEIBHOUR зеркально повторяют друг друга. Алгоритм нахождения соседнего пути можно интерпретировать как поднятие по пути от текущего узла к вершине квадрантного дерева с заменой индекса узла на каждом уровне на индекс зеркального отражения узла. Поднятие осуществляется до уровня смежности, на котором зеркальное отражение индекса узла и индекс узла не окажутся в списке дочерних индексов одной из вершин. Если уровень смежности не найден, то путь к соседнему узлу будет полностью отличаться от пути к текущему узлу, иначе, путь к соседнему узлу будет отличаться от пути к текущему узлу, начиная с уровня смежности. В работе представлена процедура алгоритма TOP_NEIBHOUR, алгоритмы LEFT_NEIBHOUR, RIGHT_NEIBHOUR и BOTTOM_NEIBHOUR не включены, так как имеют очень похожую реализацию.

Для узла, представленного на рис. 3, алгоритм GET_ADJACENT_NODE_PATHES возвращает пути [0, 3, 3] и [3, 0, 0].

```
procedure GET_AGJACENT_NODE_PATHES(PATH):
    # STACK() - вспомогательная функция, создающая пустой стек
    begin
        path_list = STACK()
        orient = PATH.pop()
        PATH.push(orient)
        if orient == 0 then
        begin
            path_list.push(TOP_NEIBHOUR(PATH))
            path_list.push(LEFT_NEIBHOUR(PATH))
        end
        else if orient == 1 then
        begin
            path_list.push(TOP_NEIBHOUR(PATH))
            path_list.push(RIGHT_NEIBHOUR(PATH))
          end
```

```
        else if orient == 2 then
        begin
            path_list.push(BOTTOM_NEIBHOUR(PATH))
            path_list.push(LEFT_NEIBHOUR(PATH))
        end
        else # orient == 3
        begin
            path_list.push(BOTTOM_NEIBHOUR(PATH))
            path_list.push(RIGHT_NEIBHOUR(PATH))
        end
        return path_list
    end

procedure TOP_NEIBHOUR(PATH):
    # LIST() - вспомогательная функция, создающая пустой список
    # MERGE() - функция объединяющая два списка
    begin
        out = LIST()
        do
            ind = PATH.pop()
            if ind == 0 then
            begin
                out.insert(0, 2)
            end
            else if ind == 1 then
            begin
                out.insert(0, 3)
            end
            else if ind == 2 then
            begin
                out.insert(0, 0)
            end
            else # ind == 3
            begin
                out.insert(0, 1)
            end
        while ind == 0 or ind == 1
        out = MERGE(PATH, out)
        return out
    end
```

Рассматриваемый алгоритм не учитывает граничные условия, т.е. текущий узел не должен быть корнем или крайней вершиной дерева. Исключение граничных условий оправдано при глубине дерева больше 8 и наличии по краям исходного изображения фоновых областей небольшого размера[*]. Результаты тестирования функций алгоритма показали, что исключение граничных условий позволяет уменьшить время выполнения на 2—3 %.

Полурекурсивный алгоритм маркировки.

Алгоритм заключается в отыскании в дереве узлов, удовлетворяющих заданным критериям, маркировке этих узлов и рекурсивного обхода всех смежных узлов. Вместо рекурсивного обхода может быть применен итеративный алгоритм обхода - поэлементный перебор всех узлов квадрантного дерева на заданном уровне и последующая маркировка. Итеративный алгоритм обладает следующими недостатками:

- Некоторые узлы квадрантного дерева могут отсутствовать на определенном уровне, если они полностью заполнены цветом фона, поэтому использование итеративного алгоритма затруднительно.
- Алгоритм перебирает все узлы дерева, что в лучшем случае приближает его по скорости выполнения к итеративному, а в среднем время выполнения алгоритма будет больше в два раза[**].

[*] При отсутствии фоновых областей по краям их легко создать, добавив ко всем пикселам смещение на один блок.

[**] При условии, что на рассматриваемом уровне не менее 50% заполнено цветом переднего плана.

• Дерево является рекурсивной структурой и не предоставляет встроенных средств для поуровневой обработки своих узлов, следовательно, итеративный алгоритм должен реализовать дополнительные функции.

• После маркировки рекурсивным алгоритмом требуется второй проход по маркированным узлам для объединения смежных областей.

Рекурсивный алгоритм не имеет ни одного из этих недостатков, но его применение также вызывает затруднения. Квадрантные деревья, хранящие большие изображения, могут содержать свыше 10 000 000 узлов, поэтому использование чисто рекурсивных алгоритмов не обоснованно ввиду очень высокой вероятности переполнения стека при работе с подобными объемами данных. Квадрантное дерево имеет очень малую высоту и большую ширину, поэтому рекурсивный обход в глубину дерева всегда будет выполняться за малое время, а обход дерева в ширину по нижним уровням посредством рекурсии часто нереализуем* за счет аппаратных ограничений. Компромиссным решением является полурекурсивный алгоритм.

Полурекурсивный алгоритм маркировки производит обход квадрантного дерева рекурсивно, а маркировку узлов дерева рекурсией через итерацию. Процедура LABEL_NEIBHOURS иллюстрирует алгоритм полурекурсивной маркировки. Список путей смежных вершин помещается в pathes, затем по каждому пути выбирается соответствующий узел, который отмечается меткой. Процедура повторяется до тех пор, пока хотя бы один узел из рассматриваемой области имеет непомеченные смежные узлы. Процедура NODE_CONDITION возвращает истину, если узел дерева удовлетворяет условию маркировки. Процедура LABEL_NEIBHOURS помещает ссылки на все узлы с меткой LABEL в список LABELED_LIST.

Полурекурсивный алгоритм маркировки имеет сложность $O(n\log(n))$, где n — количество непустых узлов квадрантного дерева.

```
procedure LABEL_NEIBHOURS(NODE, LABEL, LABELED_LIST):
    # процедура LABEL_NEIBHOURS маркирует область, начиная с узла NODE
    меткой
    # LABEL. Если маркировка была проведена успешно, процедура
    возвращает истину
    # процедура GET_PATH возвращает путь к узлу
    # процедура REMOVE_DUPLICATES удаляет из стека одинаковые значения
    # прицедура IS_EMPTY возвращает истину, если стек пуст
    # прицедура IS_MARKED возвращает истину, если узел был предварительно
    # маркирован процедурой NODE_CONDITION
    begin
        if not IS_MARKED(NODE) or NODE.label > 0 then
        begin
            return False
        end
        NODE.label = LABEL
        LABELED_LIST.add(NODE)
        pathes = GET_AGJACENT_NODE_PATHES(GET_PATH(NODE))
        stk = Stack()
        do
            foreach p in pathes do
            begin
                nd = GET_NODE_BY_PATH(p)
                if NODE_CONDITION(nd) and nd.label == 0 then
                begin
                    stk.push(nd)
                end
            end
            pathes = Stack()
            while not IS_EMPTY(stk)
            begin
                nd = stk.pop()
                nd.label = LABEL
```

* Для деревьев, имеющих глубину более 15 уровней.

```
            LABELED_LIST.add(NODE)
            foreach p in GET_AGJACENT_NODE_PATHES(GET_PATH(nd))
        end
        REMOVE_DUPLICATES(pathes)
    while not IS_EMPTY(pathes)
    return True
end
```

Кластеризация маркированных областей

Исходными данными для кластеризации являются списки блоков маркированных областей.

Маркированные области хранят в себе информацию о связных компонентах изображения. Кластеризация основана на выявлении маркированных компонентов с текстом и последующем разделении текста и графики [9, 10].

Обработка маркированных областей проводилась исходя из предположения, что области с текстом расположены вертикально или горизонтально. Для выбора областей были выявлены метрические свойства, которыми должны обладать текстовые области:

• Минимальная и максимальная высота и ширина текста в блоках. Если выбирать достаточно мелкие размеры блоков маркировки, то на области будут содержаться отдельные буквы или слова. Исходя из размеров шрифта и максимальной длины слова, можно выбрать максимальные и минимальное значения ширины и высоты области в блоках.

• Максимальная площадь текста в блоках. Минимальная площадь текста определяется исходя из минимальной высоты текста. Максимальная площадь может доходить до произведения максимальной ширины и высоты текста, но при этом, возможно, будут отбираться небольшие рисунки на исходном изображении.

• Нормированные гистограммы областей изображения по осям X и Y. Как показано в [7, 8], гистограммы областей изображения могут быть эффективно использованы для поиска текста и разделения слов на отдельные буквы. Гистограммы по осям X и Y формируют вектор признаков.

Кластеризация проводится на основе анализа метрических свойств, и, если метрические свойства удовлетворяют критериям для текстовых данных, вычисляется вектор признаков, затем, на основе его анализа, принимается решение о кластеризации. Если метрические свойства не удовлетворяют эталонным условиям для текста, область классифицируется как нетекстовая.

Для шрифтов различного размера кластеризация областей может быть проведена необходимое количество раз.

На рис. 4 приведен пример работы алгоритма*, шумы на изображении были оставлены преднамеренно.

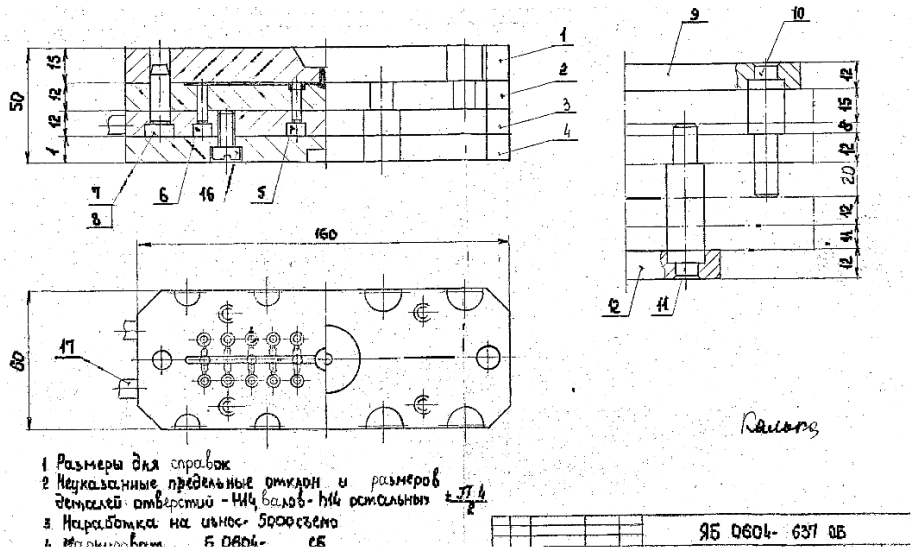

Рис. 4. Пример работы алгоритма. Выделенные текстовые компоненты отмечены насыщенным черным цветом и наложены поверх исходного изображения

* В этом примере для кластеризации текста не использовались гистограммы по осям X и Y; кластеризатор анализировал только размеры и площадь маркированных областей.

Выводы

Время выполнения алгоритма определяется временем выполнения его наиболее трудоемкой части, таким образом, время выполнения предложенного алгоритма $O(n\log(n))$.

Предложенный алгоритм маркировки имеет ограниченное применение, и он характеризуется большим расходом памяти, однако, ввиду особенностей древовидной структуры, обход изображения, представленного квадрантным деревом, происходит существенно быстрее, чем обход исходного изображения, а, благодаря свойствам квадрантного дерева сохранять изображения в разных разрешениях, данные об изображении могут быть многократно использованы как алгоритмами маркировки, так и алгоритмами кластеризации.

Алгоритмы кластеризации могут быть адаптированы для выделения объектов с любыми метрическими свойствами, при этом их сложность попрежнему будет оцениваться как $O(n\log(n))$, где n — общее количество не белых блоков в дереве, размер которых определяется самим алгоритмом кластеризации.

Алгоритм может быть усовершенствован с целью выделения любых компонентов изображения, а не только текстовой информации.

Для маркировки текстовых компонентов на изображениях с большим количеством графики разработанный алгоритм имеет большую практическую ценность. Алгоритм является альтернативой применению к изображению последовательно алгоритмов маркировки и выделения текстовых компонентов на изображении и благодаря древовидной структуре работает быстрее, чем несколько последовательных алгоритмов. Кроме того, алгоритмы на деревьях хорошо распараллеливаются.

В алгоритм возможно внести изменения для выделения областей изображения с произвольными свойствами, однако это может изменить сложностную оценку.

Библиографический список

1. *Samet H.* A Top-Down Quadtree Traversal Algorithm // Pattern Analysis and Machine Intelligence pp. 94-98, Jan. 1985.

2. *Samet H.* Connected Component Labeling Using Quadtrees // Commun. ACM. vol. 28, pp. 487-501, Jul. 1981.

3. *Jaehwa Park, Govindara ju, V., Srihari, S.N.* OCR in a hierarchical feature space // Pattern Analysis and Machine Intelligence. vol 22. pp. 400-407. Apr 2000.

4. *Samet H.* The Quadtree and velated Hierarchical Data Structures // Computing Surveys. Vol. 16. № 2. Pp. 187—260. June 1984.

5. *Shaer C A., Samet H.* An Optimal Quadtree Construction Algorithm // Proceedings of the Eighth International Conference on Pattern Recognition. pp. 317-319, Oct. 1986.

6. *Samet H.* Reconstruction of Quadtrees from Quadtree Medial Axis Transforms // Computer vision, graphics, and image processing. vol. 29. pp. 311-328, May 1985.

7. *Behera A., Lalanne D., Ingold R.* Visual Signature based Identification of Low-resolution Document Images // ACM. Pp. 178 - 187, Oct. 2004.

8. B.Anuradha SRINIVAS, Arun AGARWAL, C.Raghavendra RAO. An Overview of OCR Research in Indian Scripts // International Journal of Computer Sciences and Engineering Systems, vol.2, No.2. Pp. 141-153 Apr. 2008.

9. *Qing H. Liu, Chew Lim Tan.* Automatic Indexing of Newspaper Microfilm Images // Proceedings of the 5th International Workshop on Document Analysis Systems V. Pp. 365-375. 2002.

10. *Zhou R.W., Quek C., Ng G.S.* A novel single-pass thinning algorithm and an effective set of performance criteria // Pattern Recognition Letter. Pp. 1267-1275. Mar. 1995.