МАТЕМАТИЧЕСКОЕ МОДЕЛИРОВАНИЕ И ПРОГРАММИРОВАНИЕ

УДК 51:004.738.5+004.8

А.П. Головко
Курган, Россия

В.Н. Лапин
Санкт-Петербург, Россия

РЕПРЕЗЕНТАТИВНО-РОЛЕВАЯ МОДЕЛЬ СОДЕРЖИМОГО ВЕБ-СТРАНИЦЫ

Аннотация. Автоматический анализ содержимого (контента) веб-страниц является актуальной задачей, при этом анализ может служить самым разнообразным целям. Одной из задач, которая встает на практике, является выявление ролевой структуры контента: можно выделить основную статью, комментарии читателей, рекламу и другие функциональные роли. Решение этой задачи, кроме прочего, является важным шагом в направлении более глубокого автоматического анализа семантики страницы в дальнейшем.

Был выбран подход, согласно которому роль фрагмента определяется соответственно тому, как он внешне выглядит на экране, т.е. какова его репрезентация. Это соответствует человеческому способу восприятия.

Разработанная модель позволяет выделить фрагменты html-кода, выполняющие роли главного заголовка и основной статьи страницы. При этом основная статья может содержать разнородные элементы: текст, рисунки, таблицы и т.д., из нее удаляются внедренные фрагменты другого назначения (реклама и др.), могут использоваться различные компоновки контента на странице и способы верстки.

Модель представляет собой экспертную систему. Ее база знаний включает 1) семантическую сеть, отражающую связи между объектами и понятиями, которые используются в ходе решения задачи, 2) продукционную систему, содержащую правила, по которым осуществляется вывод. Стратегия вывода строится таким образом, чтобы исключить итерации. В ходе вывода сначала отбираются все кандидаты на определенную роль, т.е. фрагменты, которые в принципе могут оказаться в этой роли. Затем последовательно число кандидатов сокращается до одного, поскольку у ролей главного заголовка и основной статьи может быть только по одному исполнителю. Продукционная система имеет иерархическую структуру, каждая локальная система состоит из 5–10 правил и имеет автономное хранилище промежуточной информации, что сводит к минимуму возможность появления побочных эффектов.

Данная модель реализована программой на языке Python. Программа считывает html-файл из Интернета, удаляет все кроме главного заголовка и основной статьи и сохраняет результат на диске. Проверка работоспособности проводилась на новостных сайтах и на сайте Habrahabr. Результаты в каждом случае оценивались экспертно. Доля правильно обработанных страниц составила 85–90% в случаях табличной верстки и 95–97% – в случае блочной.

Ключевые слова: веб; моделирование; искусственный интеллект.

Сведения об авторах: Александр Павлович Головко[1], доцент кафедры программного обеспечения автоматизированных систем, Вадим Николаевич Лапин[2], студент 2 курса факультета информационных технологий.

Место работы: [1]Курганский государственный университет, [2]Смольный институт Российской академии образования.

Контактная информация: 649923, г. Курган, Заозерный район, 3 мкр., д. 31, кв. 84; тел.: 9129783793. E-mail: apg49@mail.ru.

1. Постановка задачи

Проблема. Автоматический анализ содержимого (контента) веб-страниц является актуальной задачей, при этом анализ может служить самым разнообразным целям.

Одной из задач, которая встает на практике, является выявление ролевой структуры контента: можно выделить основную статью, сопутствующие материалы, комментарии читателей, информацию о сайте, рекламу и другие функциональные

роли. Кроме того, сами эти единицы могут иметь достаточно развитую структуру: например, основная статья может быть монолитной или состоять из равнозначных или вложенных фрагментов, текст и фото-, видео-, аудиоматериалы могут играть основную, вспомогательную или равноправную с текстом роль и т.д.

Одна из практически востребованных задач состоит в автоматическом выделении основной статьи (включая текст и другие компоненты). Далее, например, все остальное удаляется и резко «облегченная» страница хранится в облаке или ином хранилище. Такую услугу предлагают многие сервисы, например, [1]. Весьма перспективной представляется задача автоматического семантического анализа (в первую очередь) основного содержания страницы на предмет «о чем она на самом деле» и т.п. Существующие методы служат, в основном, целям оптимизации сайтов на предмет их продвижения.

Таким образом, встает задача разработки модели веб-страницы, которая при наложении ее на конкретный HTML-файл позволит выделить фрагменты HTML-текста, соответствующие различным функциональным ролям с точки зрения посетителя страницы.

Стратегия моделирования. На первый взгляд задача должна решаться просто. Сам язык HTML предполагает разбивку страницы на иерархически вложенные разделы. Теги <h1>,…<h6> служат для обозначения заголовков этих разделов, причем с явным указанием иерархии: Для заголовка основной статьи естественно использовать <h1> и далее вниз по убывающей. Более того, стандарт HTML-5 позволяет и фактически рекомендует проводить и семантическую разметку, вводятся теги, основное предназначение которых именно в этом, такие как <section> и др. [4]. Некоторые сайты используют внутренний стандарт, согласно которому такая семантическая разметка является обязательной; это выяснено при проведении работы, о которой говорится в данной статье.

Однако, как правило, на практике дело обстоит иначе. Как показал проведенный анализ примерно 20 сайтов, в большинстве случаев HTML-текст структурирован плохо. Используемые генераторы HTML-кода пишутся так, чтобы удовлетворять сиюминутные утилитарные потребности разработчика. О какой семантической разметке может идти речь, если даже синтаксис HTML регулярно нарушается: комментарии внутри дескрипторов, отсутствие обязательных закрывающих дескрипторов или, наоборот, больше, чем открывающих, по несколько дескрипторов <head>, <body> и т.д. Веб-браузеры «прощают» это, автоматически исправляя, видимо, из соображений конкурентоспособности друг с другом.

Следовательно, простая декларативная модель, формально сопоставляющая различным блокам HTML-текста разные роли в соответствии с используемыми тегами (т.е., фактически, на основе синтаксической информации) может быть построена, но она будет иметь крайне ограниченное применение, по крайней мере, сейчас и в обозримом будущем.

Каким образом подойти к решению задачи?

Как уже упоминалось, имеется ряд программ, выполняющих очистку страницы от всего, кроме основной статьи. Прямое тестирование этих программ показало, что они действуют безукоризненно примерно в 85% случаев (тестовые примеры подбирались не тривиальные), при этом наблюдается определенная тенденция: одни программы чаще ошибаются, имея дело с таблицами, другие – с фото и видео, и т.д. Можно предположить, что в основе этих программ – движки с «навороченной» логикой, написанные в некоторой традиционной парадигме. Возможности такого подхода представляются ограниченными, особенно если говорить о более глубоком анализе семантики в дальнейшем.

Перспективным представляется другой подход. Человек – посетитель сайта визуально воспринимает интерпретированный браузером текст страницы и, обычно, легко определяет, где главное. Дизайнер проектирует страницу так, чтобы именно человеку было просто сориентироваться. Следовательно, программа-распознаватель ролей должна подражать человеку, т.е. по существу быть системой искусственного интеллекта. Такой подход одновременно готовит почву для более глубокого анализа семантики в перспективе.

Поскольку человек находит главное на странице, не перечитав все тексты и сравнив их, а по тому, как соответствующий фрагмент визуально представлен (репрезентован), соответствующую модель естественно назвать **репрезентативно-ролевой**.

Какую парадигму искусственного интеллекта выбрать? Наиболее естественными представляются экспертная система и нейронная сеть. Последняя, возможно, перспективнее. Однако на первых порах, когда не очень ясно, какие признаки учитывать, при том, что сам объект (текст длиной минимум в несколько килобайт) чрезвычайно многообразен, такое решение представляется трудно реализуемым. Поэтому выбирается экспертная система. Возможно, при накоплении материала станет реальным использование нейронной сети, в частности, комбинируя с экспертной системой. Модель, рассмотренная в данной работе, ориентирована на поиск исполнителей двух ролей: главного заголовка и основной статьи веб-страницы.

Этот вопрос мы обсудили довольно детально, поскольку, по нашему мнению, речь идет о выборе стратегии проведения исследований в данном направлении.

2. Содержательное описание задачи

Нас будут интересовать две роли: Главный заголовок и Основной текст. Причем Основной текст может содержать разнородные элементы: собственно текст (в том числе – разбитый на части каким-то образом), таблицы, рисунки. Наша задача – действовать, подражая человеку.

Как человек на взгляд определяет, где главное?

Главный заголовок – самый крупный или, по крайней мере, не мельче прочих. Он, скорее всего, имеет много общего с заголовком страницы (тэг <title>), часто просто совпадает. Наконец, он стоит выше в представлении страницы на экране, чем другие заголовки, имеющие аналогичные характеристики. Он не мерцает, не движется по экрану и т.д.

Основной текст, скорее всего, наиболее крупный фрагмент. Он занимает приблизительно центральное положение на экране. Он тесно связан с Главным заголовком. При этом «на его территории» могут находиться и фрагменты, не входящие в его состав, или даже не имеющие к нему отношения (реклама).

Здесь приходится обратить внимание на роль Комментарий. По объему текста комментарии пользователей могут значительно превосходить Основную статью. Заголовок комментария может быть не меньше главного, может почти совпадать с заголовком страницы. Текст комментариев верстальщик может поместить в один HTML-контейнер с Основной статьей. Следует отметить, что Главный заголовок не содержит слово «комментарии» в любой форме, если только этого слова нет в <title>.

Также остается проблема ошибок и нарушений синтаксиса HTML, допущенных при верстке.

3. Архитектура модели как экспертной системы

Учитывая характер задачи, была принята стратегия, аналогичная стратегии «наименьшего риска» [5. С. 153] в его регулярном варианте (без случайных откатов). Стратегия предполагает, что если некоторая подзадача в составе задачи не может быть решена полностью, ее решение пре-

рывается, мы переходим к другой подзадаче, а к этой возвращаемся, когда появится нужная информация. При этом однажды принятые решения уже не нуждаются в пересмотре.

Согласно выбранной стратегии в ходе вывода сначала отбираются все кандидаты на определенную роль, т.е. фрагменты, которые в принципе могут оказаться в этой роли. Затем последовательно число кандидатов сокращается до одного. Напомним, что в нашем случае (поиск главного заголовка и основного текста) на каждую роль может приходиться ровно один исполнитель.

Важным требованием здесь является отсутствие откатов на уровне стратегии и итерационных циклов на тактическом уровне. Все блоки алгоритма верхнего уровня работают последовательно ровно 1 раз.

На нижнем уровне обработки информации, где как раз отрабатывается много мелких деталей и где возможны частые модификации подробностей алгоритма, используются небольшие продукционные системы: по 5–10 правил. Здесь есть опасность зацикливания, так как они имеют управляющую структуру по принципу «выполнять, пока есть хотя бы одно релевантное правило». Однако при столь небольшом объеме этим моментом достаточно легко управлять, а при возникновении проблем – исправлять. Каждая такая система имеет свою локальную базу для промежуточных данных, поэтому побочные эффекты не угрожают. Наконец, в перспективе это может пригодиться для распараллеливания процесса, так как продукционная система обладает естественным параллелизмом [2. С. 53].

Практически все блоки промежуточной информации имеют структуру списков. Для их обработки используются программные средства обработки именно списков, а не массивов, что исключает зацикливание.

Таким образом, работа системы последовательно декомпонирована на мелкие, достаточно независимые фрагменты. Платой за это являются многочисленные и большие по объему массивы промежуточной информации. Они, однако, имеют несложную структуру и четко различаются по назначению, что облегчает задачу управления.

4. Информационный компонент модели

На уровне модели ее информационная составляющая представлена семантической сетью, которая включает 2 компонента: статический и динамический. Статический содержит понятия, которые используются в процессе вывода и не зависят от конкретной решаемой в данный момент задачи. Динамический отражает именно каждую конкретную задачу (веб-страницу). В целях удобочитаемости эта часть системы представлена в несколько другом виде. Понятия и связи из статической части даны в виде определений, причем опущены те, смысл которых итак понятен; из динамической части – в основном, в виде списков, идентификаторы которых соответствуют коду программы. Например, список TxtListIdsPre – «Список вообще всех ET, непосредственно содержащих текст», на языке семантической сети означает набор экземпляров связи «Является», соединяющих некоторое множество элементов текста страницы с понятием «Блок, содержащий текст» и т.д.

4.1. Статическая информация (понятия)

1. Сам текст кода представлен обычной строкой символов SourHtml. Его элементы (обычные Юникод-символы) будем нумеровать, начиная с нуля.

2. Элемент текста ET – фрагмент html-кода веб-страницы, который

1) включает в себя некоторый открывающий дескриптор, соответствующий ему закрывающий дескриптор и текст между ними. Если практически закрывающий дескриптор опущен, но ясно, где он подразумевается (например, <p>), то ET включает фрагмент от начала открывающего де-

скриптора до точки, где подразумевается закрывающий;

2) если закрывающий дескриптор не положен по синтаксису языка (например,
), то включает только текст открывающего дескриптора.

Если X – некоторый ET, то beg(X) и end(X) – это, соответственно, номера позиций в SourHtml, соответствующих первому ('<') и последнему ('>') символам, принадлежащих X, а type(X) – тип открывающего дескриптора, с которого начинается этот фрагмент кода, например, .

Два ET могут либо не пересекаться, либо один может быть вложен в другой.

3. AllET – множество всех ET блокового уровня [4], которые могут быть построены для данной SourHtml. Теги <html>, <head>, <body> относим сюда же.

4. Основной граф TreeOfElems – граф (дерево), представляющий структуру множества AllET. Он является базовым информационным объектом в алгоритме (не считая самой SourHtml). Имеет следующую характеристику

1) В корне дерева — фиктивный элемент «Все дерево». Дальше — по вложенности.

2) Дерево TreeOfElems представлено вектором (массивом). При этом

– каждый элемент представляет один ET (**не** дескриптор);

– расположение элементов TreeOfElems всегда соответствует расположению ET в тексте. То есть, если открывающий дескриптор ET ***A*** расположен в SourHtml раньше, чем такой дескриптор ET ***B***, то индекс ET ***A*** в TreeOfElems будет меньше, чем индекс ET ***B***;

– любое поддерево всегда представляет собой сплошной фрагмент TreeOfElems, первый элемент фрагмента — корень поддерева.

3) Каждый элемент имеет уникальный идентификатор – целое число, начиная с нуля. Первоначально (при построении дерева) идентификатор равен индексу в TreeOfElems. Затем, при преобразованиях дерева, индекс может изменяться, идентификатор сохраняется. Все требования предыдущего пункта также сохраняют силу при преобразованиях дерева.

4) Каждый элемент TreeOfElems представлен кортежем, включающим информацию, позволяющую определить как положение в дереве, так и по отношению к SourHtml.

5. Текстовая таблица TblTxt – таблица (ET <table>), служащая для представления данных: текста, рисунков и т.д. Может содержать вложенные текстовые таблицы.

6. Организующая таблица TblOrg – таблица (ET <table>), служащая для организации нужного расположения фрагментов страницы на экране. Используется при табличной верстке. Может содержать вложенные организующие и текстовые таблицы.

7. Элемент, содержащий текст, – TxET – это ET, включающий

– хотя бы один ET типов <p>, <pre>, <blockquote> – непосредственно

и / или

– хотя бы один ET типа <img> и/или текстовую таблицу — непосредственно или не непосредственно.

8. Путь к элементу PhET – путь от корня TreeOfElems к какому-то фрагменту.

PhET строятся для ET либо отдельных подстрок текста, полностью входящих в состав одного ET — листа дерева, в частности — для отдельных символов.

PhET представляет собой список, в начале которого стоит корень всего TreeOfElems, а в конце — ET, для которого этот PhET построен, или тот ET, в состав которого входит подстрока, для которой строится матрешка. Каждый элемент PhET представлен парой (идентификатор элемента дерева, его индекс в дереве).

9. Марк-лист MrkLst. Подобен PhET, однако последний элемент этого списка — список Sp: не пустой, но возможно, состоящий из одного элемента. Каждый элемент Sp задает один элемент TreeOfElems тем же способом, что и в PhET. Все элементы Sp

должны являться непосредственными потомками одного и того же элемента TreeOfElems. Список Sp будем называть хвостом марк-листа, а остальную часть — головой. Практическое назначение марк-листов – пометка фрагментов, детектированных на определенную роль.

10. Корневой предок RootPar (для ET типов <h1>,...,<h6>) – ET, являющийся самым близким предком данного ET и являющийся TxET.

11. Гроздь заголовков BrHead (здесь «заголовок» – ET любого из типов <h1>, ...,<h6>). Гроздь – это минимальное поддерево в дереве TreeOfElems, в которое входят все эти заголовки. Гроздь содержит не менее двух заголовков.

12. Используются предикаты: Anc(X, Y) – X является предком для Y в TreeOfElems, Par(X, Y) – X является родителем для Y в TreeOfElems, DescrBody(X) – X является телом некоторого дескриптора.

4.2. Динамическая информация (объекты)

В таблице 1 представлены основные информационные объекты, порождаемые при моделировании одной веб-страницы.

Таблица 1

Основные динамические информационные объекты модели

№	Идентификатор	Семантика
1	SourHtml	Исходный html-код
2	TreeOfElems	Дерево элементов текста (ET)
3	TableRgstr	Реестр таблиц
4	Hdrs	Найденные заголовки, подходящие по совпадению с <title>
5	HdrsMatr	Информация о взаимном расположении кандидатов в Главный заголовок
6	BestHeaderId	Идентификатор лучшего кандидата в Главный заголовок
7	TxtListIdsPre	Список вообще всех ET, непосредственно содержащих текст
8	TxtListIdInds	Найденные ET с текстом – первичные
9	CandCommentWordsPos	Список слов в SourHtml, похожих на «Comment» на разных языках
10	PrepairedCandCommentWord	CandCommentWordsPosr, подготовленные к рейтингу
11	BestCandCommentHeader_Id	Из кандидатов в заголовки Комментария – самый вероятный
12	CandMainTxt_Ids	Список кандидатов в Основную статью, учитывая заголовок Комментария
13	TblForCandMainTxt_Ids	Для каждого кандидата в Основную статью – список текстовых таблиц, которые в него входят непосредственно, а если не непосредственно, то не через другую текстовую таблицу
14	MatrBlkWithLongText	PhET блока с самым длинным текстом. Ищется в пределах, ограниченных HdrsMatr
15		NessTextIds. Список ET в составе блока с самым длинным текстом, которые следует включить в марк- лист
16	QuUnusTextIds	ET в составе блока с самым длинным текстом, которые, возможно, не следует включить в марк-лист
17	MarkList1BestHdr	Марк-лист главного заголовка
18	MarkList1MaxText	Марк-лист самого длинного текста
19	MarkList1Head	Марк-лист для <head> страницы
20	MarkListsTxtTables	Список марк-листов текстовых таблиц
21	MarkListsImgs	Список марк-листов рисунков
22	CommParentHeadTxtId	Ближайший общий предок Главного заголовка и Основной статьи

5. Процедурный компонент модели

Основные этапы/блоки алгоритма приведены в таблице 2. В каждом случае указаны только выходные данные. Для Blk0 входными данными является SourHtml. Остальные используют (почти) всю имеющуюся на момент их запуска информацию. После таблицы содержится очень краткое описание работы основных блоков алгоритма.

Таблица 2

Основные блоки алгоритма

№	Идентификатор блока	Назначение	Что вычисляет/ модифицирует
1	Blk0	Предварительная обработкам Простейшая обработка Реестра таблиц	TreeOfElems, TableRgstr
2	FindHeaders	Определение множества заголовков – кандидатов в Главный заголовок статьи	Hdrs, HdrsMatr
3	FindBestHdr	Определение Главного заголовка	BestHeaderId
4	FindAllTexts	Находит вообще все текстовые блоки	TxtListIdsPre
5	TxtLstCorr	Корректирует списки текстовых фрагментов (TxtListIdsPre) в связи с тем, что некоторые из них являются элементами текстовых таблиц	TxtListIdInds
6	FindCommentWords	Находит вхождения всех слов типа «Comment» в текст	CandCommentWordsPos
7	PrepairForRating-CandCommHdrs	Для каждой позиции CandComment-WordsPos находит очищенную строку. Определяет, не включена ли она в {<h1, ...,<h6> или <headre>	PrepairedCandCommentWord
8	RatingCandCommHdrs	Устанавливает, кто из кандидатов в заголовки Комментария – самый вероятный	BestCandCommentHdr_Id
9	MakeCandMainTxtList	Формирует список кандидатов в Основную статью (идентификаторы), учитывая заголовок Комментария	CandMainTxt_Ids
10	OpredTablesForCand	Для каждого кандидата в Основную статью определяет список текстовых таблиц, входящих в него	TblForCandMainTxt_Ids
11	FindMaxText	Определяет блок с самым длинным текстом и возвращает его PhET	MatrBlkWithLongText
12	PrepairToMarkingHeader AndText	По уже имеющимся и BestHeaderId MatrBlkWithLongText готовит марк-листы для заголовка и текста	MarkList1MaxText
13	PrepToMarkingHtmlHead	Готовит HEAD страницы к удалению «лишнего»	MrkLst1Head
14	PrepairToMarkingTxtTbl	По уже имеющимся QuUnusTextIds готовит марк-листы для текстовых таблиц	MarkListsTxtTables
15	PrepairToMarkingImgs	По уже имеющимся QuUnusTextIds готовит марк-листы для <img>	MarkListsImg
16	DeleteByDictLst	Объединяет все марк-листы в один. Удаляет все ET, не включенные в марк-листы	TreeOfElems[1], SourHtml[1]

Таблица 2. Продолжение

№	Идентификатор блока	Назначение	Что вычисляет/ модифицирует
17	CommParentHeadText	Находит ближайшего общего предка заголовка и текста	CommParentHeadTxtId
18	DeleteAttrByList	Из всех потомков CommParentHeadTxtId удаляет атрибут href вместе со значением	SourHtml[2]

[1]TreeOfElems, из которого удалены ET, не включенные в марк-листы.
SourHtml, в которой произведены удаления в соответствии с изменениями в TreeOfElems
[2]SourHtml, измененная в соответствии с выполненным действием.

Предварительная обработка Blk0. Из SourHtml удаляются все скрипты и комментарии. Строится TreeOfElems. Формально операция чисто техническая. Сканируется текст, выделяются дескрипторы, в TreeOfElems добавляются элементы, в них записывается нужная информация. Практически именно здесь выполняется работа по исправлению синтаксических ошибок в тексте и нередко – разгадывание загадок: что имел в виду автор. Кроме того, текст приводится к виду, требуемому для TreeOfElems. Например, если встречается «бесхозный» текст, для него создается тег <p>, и т.д. Так же строится реестр таблиц.

Определение множества кандидатов в Главный заголовок статьи FindHeaders.

1. На первом шаге алгоритма формируется TitleLower: символьная строка, получаемая следующим образом. Получается строка sTitle — тело ET <title>. К sTitle применяется процедура PreaireString, которая «чистит» строку: убирает теги строкового уровня, лишние пробелы, каждый спецсимвол заменяется обычным Юникод-символом и т.д.

2. Формируется список Hdrs1, состоящий из идентификаторов всех элементов текста X, для которых type(X) ∈ {<h1>, …, <h6>}.

3. Для каждого элемента Hdrs1 определяется степень его сходства с TitleLower, т.е. по смыслу, степень сходства данного ET с заголовком страницы. Практически ищется самая длинная совпадающая подстрока. В дальнейшем может использоваться более интеллектуальная процедура.

4. Для всей совокупности элементов Hdrs1 строится гроздь HdrsMatr – структура типа BrHead. С ее помощью можно отследить взаимное расположение кандидатов в тексте.

Определение Главного заголовка FindBestHdr. Если Hdrs содержит только один элемент, то этот кандидат (единственный) и есть Главный заголовок. Если кандидатов несколько, то они рейтингуются определенным образом. Мы считаем, что самый важный признак — тип ЭТ, содержащий текст заголовка. Реально у настоящего Главного заголовка это практически всегда <h1>. Далее, конечно, важна длина подстроки, по которой совпадает текст кандидата с текстом, содержащимся в <title>. А при совпадении (у двух и более конкурентов) обоих признаков лучше считается тот, что встречается раньше по тексту. Эту операцию производит специальная функция.

Нахождение всех текстовых ET. Просматривается весь ET <body>. Отбираются все ET, отвечающие условию: ET(X) & (∃Y)(ET(Y) & Par(X, Y) & ((type(Y) ∈ {<p>, <pre?, < lockquote >} ∨(TblTxt(Y) & ¬(∃Z)(TblTxt(Z) & (Y ⊂ Z)))), другими словами, отбираются элементы, про каждый из которых можно сказать, что он является родителем (только непосредственным, а не вообще предком) хотя бы одного ET, который отвечает одному из следующих условий: 1) это ET <p> или <pre> или <blockquote>; 2) это тексто-

вая таблица, не вложенная в другую текстовую. В дальнейшем мы будем чаще использовать содержательные, а не формальные описания.

Формируем список кандидатов в заголовок комментариев: FindCommentWords. Принцип состоит в следующем. Заголовок комментариев может состоять из одного слова «Комментарии» или то же на других языках. Заголовок может являться предложением типа «Комментарии по теме», при этом дальше может следовать и название темы, тогда заголовок комментариев будет содержать и заголовок статьи. С другой стороны, о каких-то комментариях может говориться в тексте основной статьи и еще где угодно. Необходимо найти фрагмент – самый вероятный кандидат на роль заголовка комментариев.

Имеется список CNS_CommenWords слов и фрагментов слов на разных языках. Для каждого элемента CNS_CommenWords составляется список его вхождений в SourHtml, для многих эти списки будут пусты. Далее удаляются все вхождения X, для которых выполняется DescrBody(Y) & (X ⊂ Y) ∨ (type(Z) = <a>) & Anc(Z, X)).

Дополнительно обрабатывается список кандидатов в заголовки комментариев: PrepairForRatingCandCommHdrs. Для каждого кандидата определяется его ближайший предок – ET блокового уровня. Его содержимое обрабатывается процедурой PreaireString.

Определение, кто из кандидатов в заголовки Комментария – самый вероятный: RatingCandCommHdrs. Считается, что если текст состоит из одного слова «Комментарии», «Comments» и т.п. на других языках, то это и есть самый важный признак того, что мы имеем заголовок комментариев. Эти слова составляют список CommentWholeHeader1. Следующим по важности считается «ранг» ET, в котором данный заголовок расположен. Самый приоритетный<h1>, далее <h2>,...<h6>, <header>, любой другой. И далее, при прочих равных условиях, более вероятным считается тот, кто расположен выше по тексту.

Формируется список кандидатов в основной текст: MakeCandMainTxtList. Формируется CandMainTxt_Ids – список кандидатов на роль основного текста, учитывая заголовок Комментария. Сейчас мы просто не включаем в список кандидатов тех, кто стоит в дереве после BestCandCommentHeader_Id. (Иногда заголовок Комментария оказывается выше Главного заголовка, хотя сами комментарии – всегда ниже. Этот случай обрабатывается отдельно, здесь его не рассматриваем). Практически если идентификатор элемента этого списка меньше BestCandCommentHeader_Id (или заголовка Комментария вообще нет), то элемент включается в число кандидатов.

Для каждого кандидата в основной текст составляется список текстовых таблиц, входящих в него: OpredTablesForCand. Говоря просто, если какая-то не вложенная текстовая таблица содержится в некотором контейнере, являющемся кандидатом в основной текст статьи, то этот факт фиксируется. Список Res1 – собрание таких фактов. Он вполне может оказаться пустым, если основная статья не содержит таблиц или их вообще нет. На основе этого списка для каждого кандидата формируется список входящих в него текстовых таблиц, причем не вложенных.

Определяется блок с самым длинным текстом: FindMaxText. Определяется длина текста в каждом кандидате в Основной текст статьи.

1. Для кандидата выполняется следующее:

1) Определяется длина «обычного» (т.е. вне таблиц) текста в кандидате. Для этого сканируется список потомков кандидата. Если тип потомка – один из {<p>, <pre>, <blockquote>, <ol>, }, то с помощью процедуры CalcPureTxtLen подсчи-

тывается «чистая» длина текста в этом потомке.

2) Определяется длина текста в таблицах, входящих в состав кандидата, используя TblForCandMainTxt_Ids. В каждом случае выбрасываются входящие дескрипторы любых тегов, в том числе вложенные. Отбрасываются ведущие и хвостовые пробелы. Возвращает длину полученной строки.

3) Все длины текстов суммируются.

2. Выбирается кандидат с наибольшим значением показателя. Это и есть Основная статья.

Подготовка марк-листов для Главного заголовка, Основной статьи и <head>'а html-страницы. Это заключительная стадия алгоритма по собственно распознаванию ролей, включающая несколько блоков таблицы 2.

1. Мы знаем ET с текстом Основной статьи. Следует ли этот ET включать в результат в полном составе? В общем случае – нет. Там могут быть посторонние объекты, например, реклама, ссылки на дополнительные материалы по теме и т.д. Кроме того, в этот ЭТ могут быть включены и комментарии. Мы их должны взять на заметку (для распознавания и этих ролей в будущем), но пока исключить. Поэтому сначала мы отбираем то, что заведомо нужно. Это {X | type(X) ∈ {<p>, <pre>, <blockquote>, <ol>,<ul>,<h1>,..., <h6>} }. Все остальные пока считаются предположительно ненужными. Получаются, соответственно, 2 списка NessTextIds и QuUnusTextIds. Далее из этих списков удаляем все элементы, имеющие идентификатор больше BestCandCommentHeader_Id, т.е., по-видимому, уже относящиеся к комментариям. Для элементов из NessTextIds формируем марк-лист.

2. Далее формируем MarkList1BestHdr – марк-лист для главного заголовка.

3. В ET <head> нам нужно сохранить <title> и те ЭТ <meta>, в которых есть атрибуты 'charset' и 'http-equiv'. Двух последних может и не оказаться. Мы считаем, что эти ET входят в <title> непосредственно. Их обнаружение и составление марк-листов не вызывает особенных трудностей.

4. Создается марк-лист для таблиц, которые необходимо включить в Основную статью. Для этого сканируется список предположительно ненужных элементов QuUnusTextIds. Если элемент – не вложенная текстовая таблица, то он признается годным и на него составляется марк-лист.

5. Аналогичная процедура выполняется для рисунков (ET <img>) в составе Основной статьи.

Удаление всех ET, не включенных в марк-листы, является технической операцией.

Нахождение ближайшего общего предка Главного заголовка и Основной статьи необходимо для обеспечения правильного отображения страницы на экране после всех модификаций.

6. Программная реализация

Данная модель реализована программно.

На вход программы подается список URL. Программа считывает html-файл по одному из Интернета, удаляет все кроме главного заголовка и основной статьи и сохраняет результат на диске, там он может быть просмотрен с помощью любого браузера. Пользовательского интерфейса фактически нет.

Программа написана на языке Python. Архитектура программы следующая.

Два основных компонента программы: 1) экспертная система, реализующая модель, 2) интерфейс с Интернетом и хранилищем на диске. Развитой базы данных и пользовательского интерфейса фактически нет.

Основным компонентом является экспертная система. Программно она представляет собой набор следующих основных компонентов:

1) Библиотека из примерно 35 мелких функций.

2) Хранилище глобальной информации (см. табл. 1).

3) Класс ProdSyst, реализующий продукционную систему. В ходе работы программы создается некоторое количество экземпляров класса, которые при создании инициализируются набором правил. Каждое правило представляет собой кортеж из двух небольших функций (обычно – в виде лямбда-функций): антецедента и консеквента продукции. Антецедент возвращает логическое значение и содержит, в основном, ссылки на глобальную информацию и на локальную информацию (свойства) данного экземпляра ProdSyst. Консеквент, в основном, содержит вызовы библиотечных функций. Библиотечные функции, как правило, работают только с глобальной информацией, причем узко специализированы. Язык Python здесь очень удобен, так как позволяет действовать в стиле функционального программирования, при этом каждой функцией (или набором) можно манипулировать, как одним объектом.

4) Функция, которая запускает всю систему и организует работу в целом.

В начале работы создается объект класса ProdSyst, которому передается система правил, фактически, без антецедентов, а консеквенты реализуют как раз действия, описанные в табл. 2. Те, в свою очередь, создают экземпляры ProdSyst для решения локальных задач и т.д. Этот объект существует до конца работы и разрушается последним.

7. Экспериментальная проверка

Проверка работоспособности проводилась на новостных сайтах, сайтах IT-компаний и на сайте habrahabr.ru. Сайты русско- и англоязычные. Результаты в каждом случае оценивались экспертно. Естественно, БЗ многократно дорабатывалась. В конце процесса работы над системой доля правильно обработанных страниц составила 85–90% в случаях табличной верстки и 95–97% – при блочной. Разницу можно объяснить, с одной стороны, тем, что в случае блочной верстки программа чувствительнее к ошибкам в HTML-коде, их сложнее исправлять. С другой стороны, сама модель нуждается в доработке в части распознавания HTML-страницы, созданной с помощью табличной верстки.

8. Выводы

Разработана модель содержимого (контента) веб-страницы, позволяющая выделить фрагменты html-кода, выполняющие роли главного заголовка и основной статьи страницы. При этом основная статья может содержать разнородные элементы: текст, рисунки, таблицы и т.д., из нее удаляются внедренные фрагменты другого назначения (реклама и др.), могут использоваться различные компоновки контента на странице и способы верстки.

Модель основана на подражании человеческому способу восприятия информации «на взгляд» при выделении главных фрагментов содержания, т.е. ориентируется на то, как представлена информация.

Технически модель представляет собой экспертную систему.

Модель программно реализована. Экспериментальная проверка показала достаточно высокую адекватность модели.

ЛИТЕРАТУРА

1. Приложение для сохранения информации в облаке. Pocket URL: http://getpocket.com.
2. Продукционные модели // Искусственный интеллект: В 3 кн. Кн. 2. Модели и методы: Справочник / Под. ред. Д.А.Поспелова. – М., 1990.
3. Семантика в HTML 5. URL: http://habrahabr.ru/post/49734.
4. Учебник HTML. URL: http://ru.html.net/tutorials/html.
5. Элти Дж., Кумбс М. Экспертные системы: концепции и примеры. – М., 1977.

REFERENCES

1. Prilozhenie dlya sokhraneniya informatsii v oblake Pocket [Application for storing date in Pocket cloud]. URL: http://getpocket.com. (In Russian).
2. Produktsionnye modeli [Production models]// In: Iskusstvennyj intellekt. Kniga 2. Modeli i metody: Spravochnik/ Ed. by D.A. Pospelov. Moscow, 1990. (In Russian).
3. Semantika v HTML 5 [Semantics in HTML 5]. URL: http://habrahabr.ru/post/49734/ (In Russian).
4. Uchebnik HTML [A HTML guide]. URL: http://ru.html.net/tutorials/html/ (In Russian).
5. Elti, Dj., Cumbs, M. Ekspertnye sistemy: konceptsii i primery [Expert systems: concepts and examples]. Moscow, 1977. (In Russian).

A.P. Golovko
Kurgan, Russia

V.N. Lapin
Saint-Petersburg, Russia

REPRESENTATIVE AND ROLE MODEL OF THE WEB PAGE CONTENT

Abstract. Today automatic analysis of the web page content is a topical problem. The analysis enables us to solve several practical problems, including detecting the role structure of a page content. Here we can distinguish the main page article, comments of website visitors, advertisements, and other functions. In addition, solving this problem is an important step towards a more profound automatic analysis of website semantic in the future.

We have applied the approach defining the role of some html-code fragment in accordance with the way it is represented on the screen, which corresponds to the human way of perception.

The developed model allows us to distinguish such html-code fragments acting as the main header and the main article of a page. The main article may contain different elements, such as a text, tables, images, etc. Often other elements (advertisements etc.) are deleted from the main article, and various ways of placing content elements on the screen and page layouts may be applied.

The model is an expert system with the knowledge base containing 1) a semantic net reflecting relations between objects and concepts used in problem-solving; 2) a production system containing a set of rules for the inference. The inference strategy is constructed so to exclude any iteration. During the inference, all elements that can play this role are selected, after which the number of them gradually decreases to one. The production system has a hierarchical structure, with each local system consisting of 5–10 rules and having its own local data storage, which allows us to minimize the probability of side effects.

This model is implemented as a program using Python programming language. The program reads html-file from the Internet, removes all elements except the main header and main article, and stores the result as a file on a hard disc. The program was tested on news-sites and habrahabr.ru. The proportion of correctly processed pages was 85-90% in case of the table layout of a page and 95–97% when a page was developed as a block.

Key words: web; modeling; artificial intelligence.

About the authors: Alexandr Pavlovich Golovko[1], Associate Professor at the Department of Automated Systems Software, Vadim Nickolaevich Lapin[2], 2nd year student of the Faculty of Information Technologies.

Place of employment: [1]Kurgan State University, [2]Smolny University of the Russian Academy of Education.