Труды Карельского научного центра РАН
№ 8. 2016. С. 81–90
DOI: 10.17076/mat381

УДК 004.738.5:51-7

РАЗРАБОТКА ПРОГРАММЫ СБОРА ДАННЫХ О СТРУКТУРЕ ВЕБ-САЙТОВ

А. А. Печников[1], А. В. Ланкин[2]

[1] *Институт прикладных математических исследований Карельского научного центра РАН*
[2] *Санкт-Петербургский государственный университет*

Наиболее распространенной математической моделью веб-сайта является веб-граф. Для построения веб-графа реального сайта требуются сведения о его структуре: html-страницах и/или документах сайта (в частности, об URL-адресах веб-ресурсов) и связывающих их гиперссылках. Веб-серверы часто используют псевдонимы и перенаправления, а также динамически генерируют одни и те же страницы по разным URL-запросам. Отсюда возникает проблема, которая заключается в наличии различных URL, имеющих один и тот же контент. Таким образом, мы можем получить веб-граф, у которого отдельные вершины соответствуют страницам сайта с одним и тем же контентом. В работе описывается поисковый робот (краулер) RCCrawler, основной задачей которого является сбор информации о веб-сайтах для построения их веб-графов, во многом решающий указанную проблему, что подтверждается проведенной серией экспериментов.

К л ю ч е в ы е с л о в а: веб-сайт; гиперссылка; краулер; веб-граф.

A. A. Pechnikov, A. V. Lankin. DEVELOPMENT OF A PROGRAM FOR COLLECTION OF WEBSITE STRUCTURE DATA

The web graph is the most common mathematical model of a website. Constructing a web graph of a real site requires data about the structure of that site: html-pages and/or documents in the site (in particular, data about URLs of web resources) and hyperlinks linking them. Web servers often use pseudonyms and redirections. They also generate the same pages dynamically via different URL requests. This raises a problem in which there are various URLs but with the same content. Thus, we can get a web graph in which some of its vertices correspond to pages of a site with the same content. The paper describes a crawler called RCCrawler that collects information about websites to build the web graphs of these sites. This crawler largely addresses the above problem as confirmed by a series of experiments conducted.

K e y w o r d s: website; hyperlink; crawler; web-graph.

Введение и основные понятия

Веб-сайт – совокупность html-страниц и веб-документов, связанных внутренними гиперссылками и обладающих единством содержания, идентифицируемая в Вебе по уникальному доменному имени. В настоящее время

структура веб-сайтов стала достаточно сложной и может быть сравнима с фрагментами Веба.

У веб-сайта всегда можно выделить его главную страницу, идентифицируемую по доменному имени, уровень которой будем считать равным 0. Тогда уровень любой другой страницы сайта – это минимальное количество кликов, требуемых для перехода на нее с главной страницы.

Для описания такой структуры можно использовать веб-граф сайта – ориентированный граф, вершинами которого являются html-страницы и/или документы, а дугами – гиперссылки между ними. Здесь уровень вершины легко определяется через длину кратчайшего пути до нее из начальной вершины.

Веб-графы как объекты исследований рассматриваются в качестве моделей сайтов во многих работах. К примеру, в [2, 12] веб-граф сайта используется в качестве модели для описания поведения пользователя при перемещении по сайту (так называемого «серфинга»). В [7] с использованием веб-графа дается оценка максимальной глубины сканирования веб-сайта для поиска требуемой информации. Некоторые свойства веб-графа сайта, а именно наличие выделенной начальной вершины, определение уровневой структуры через удаленность веб-страниц от главной страницы и почти всегда сильная связность формулируются в [4].

Для построения веб-графа реального сайта требуются сведения о его структуре: html-страницах и/или документах сайта (в частности, URL-адреса веб-ресурса) и связывающих их гиперссылках, и для этого нужна программа-краулер, умеющая извлекать такие сведения из Веба. Термин «краулер» (англ. crawler) в общем случае обозначает программу, реализующую процесс перемещения по страницам и/или документам Веба с целью сбора определенной информации, статистики или сохранения ресурсов сайта. Иногда в статье такой процесс будем называть краулингом (англ. crawling). Общим принципам разработки краулеров посвящена работа [13].

URL (Uniform Resource Locator) представляет собой стандартизированный способ записи адреса веб-ресурса (html-страницы, изображения, файлы, медиа-потоки и т. д.) в Вебе [15,16]. Гиперссылки (или просто ссылки) являются базовыми элементами Веба, связывающими веб-ресурсы [1]. В простейшем виде можно считать, что гиперссылка – это пара <URL-источник, URL-мишень>, где URL-источник является адресом веб-ресурса, с которого сделана ссылка, реализующая возможность перехода на веб-ресурс с адресом URL-мишень. В частности, гиперссылки, соединяющие страницы и документы одного сайта, задают его внутреннюю структуру; мы будем называть их внутренними гиперссылками в отличие от гиперссылок, соединяющих страницы разных сайтов (то есть внешних гиперссылок).

В статье [17] обсуждаются вопросы, когда различные URL отсылают к одинаковому контенту. Такие повторяющиеся URL-адреса широко распространены на веб-сайтах, так как программное обеспечение веб-сервера часто использует псевдонимы и перенаправления (редиректы), приводя URL-адреса к некоторой канонической форме, и динамически генерирует ту же самую страницу по различным URL.

Например, несложно проверить, что на сайте факультета прикладной математики – процессов управления Санкт-Петербургского государственного университета три разных URL
http://www.apmath.spbu.ru,
http://www.apmath.spbu.ru/ru/
и http://www.apmath.spbu.ru/index.html
отсылают к одной и той же (начальной) странице.

Еще один пример с сайта Института прикладных математических исследований КарНЦ РАН – два разных URL страницы лаборатории телекоммуникационных систем:
http://mathem.krc.karelia.ru/
structure.php?plang=r&id=P8 и
http://mathem.krc.karelia.ru/
structure.php?id=P8&plang=r.

Принципиальная возможность избегания разных URL с одинаковым контекстом заключается в сравнении эскизов получаемых страниц/документов, например, по частотному распределению встречающихся слов. Поскольку эта процедура может проводиться только постраничным сравнением, то сравниваемые страницы должны быть предварительно извлечены, что требует очевидных затрат ресурсов, которых хотелось бы избежать, особенно для сайтов, содержащих десятки тысяч страниц.

Часть вопросов, возникающих в связи с повторяющимся контентом, снимается нормализацией URL – процессом, при котором URL приводится к единообразному виду для того, чтобы определить эквивалентность двух синтаксически различных URL-адресов [16].

Проблема «повторяющихся URL», очевидно, может приводить, во-первых, к лишним затратам времени, а во-вторых, к многократно-

му повторению получаемых данных. Возможно, она не представляет больших неприятностей для случая, например, поиска внешних гиперссылок при исчерпывающем поиске на сайте: совпадающие ссылки можно отсеять на завершающем этапе работы.

Но в нашем случае она приводит к построению веб-графов сайтов, имеющих дубли вершин и ссылок, то есть получению неадекватных математических моделей, ведущих к существенным погрешностям при их анализе, а значит, и к неправильным результатам и выводам.

Несмотря на серьезное продвижение в разработке краулеров за последние 5-6 лет (краткий список свободно распространяемых программ можно посмотреть здесь [20]), нам не удалось найти краулер, основной задачей которого является сбор данных о страницах и внутренних ссылках сайта. Практически во всех случаях перечень страниц является побочным эффектом работы краулера, а вопрос о сохранении данных по всем внутренним гиперссылкам не стоит совсем. На практике краулер может «забывать» гиперссылки, по которым он перемещается, важно хранить список страниц и ссылок, по которым он попал на эти страницы.

Отсюда следует основная цель данной работы, заключающаяся в разработке специализированного краулера, решающего задачу сканирования заданного веб-сайта с целью сбора данных о его внутренней структуре, используемых далее для построения веб-графа сайта, с дополнительным требованием минимизации проблемы повторяющихся URL.

Краулер RCCrawler: требования к разработке

В данном разделе описаны основные и дополнительные требования, в соответствии с которыми был разработан краулер, названный Rapid Configurable Crawler (RCCrawler).

Основные требования к программе:

1. Получать в качестве исходных данных список доменных имен сайтов, предназначенных для сканирования. Предполагается, что имеется некоторое множество исследуемых сайтов, поэтому должна быть предусмотрена возможность сканирования нескольких сайтов одновременно.
2. Обходить каждый сайт, начиная с главной (индексной) страницы, перемещаясь по внутренним гиперссылкам в заданном порядке обхода «вначале вширь» [3].
3. Выдавать в качестве результата данные для построения веб-графов сайтов, полученных на входе, в виде файлов, имена которых ассоциируемы с доменными именами сайтов.
4. Иметь возможность обхода нескольких сайтов одновременно.
5. Иметь расширяемую архитектуру для последующего развития функциональности.
6. Иметь механизмы внешней остановки работы приложения с сохранением состояния для продолжения работы с точки останова.
7. Иметь механизмы внутренней остановки работы приложения в специально перечисленных случаях типа: не осталось не обойденных внутренних гиперссылок, по истечении времени ожидания доступа к веб-ресурсу, по достижении заданной глубины сканирования.
8. Следовать правилам из файла ограничения доступа к содержимому сайта robots.txt [5], что помогает избегать ненужных ссылок вроде страниц поиска, авторизации, сортировки по заданным параметрам, приводящих в некоторых случаях к бесконечному процессу сканирования.

Дополнительные требования:

1. Объектами сканирования являются только html-страницы, а гиперссылки, указывающие на файлы с расширениями rar, ppt, xls, doc, js и прочие, не рассматриваются.
2. Гиперссылки извлекаются из полученной html-страницы из тегов <a> без предварительного выполнения сценариев и имитации действий пользователя. Такое допущение упрощает разработку и позволяет меньше нагружать ресурсы компьютера.
3. Для гиперссылки, содержащей URL, соответствующей узлу веб-графа, сервер должен выдавать ответ с кодом состояния HTTP [19], равным 200 (OK – успешный запрос).
4. В случае, если код ответа равен 301 (Moved Permanently), 302 (Moved Temporarily) или 303 (See Other) [11], необходимо выполнить переход по указанному в ответе URL и заменить адрес обрабатываемой страницы новым, по которому выполняется переход. Такое требование связано с тем, что пользователь в окне браузера получает страницу по конечной ссылке и «не замечает» промежуточных страниц с переходами (редиректами).
5. Следует игнорировать внешние гиперссылки, включая случаи, когда это ссылка на поддомен данного домена. В рамках данной работы поддомен считается отдельным сайтом в соответствии с данным ранее определением сайта.
6. Следует игнорировать ссылки с протоколом передачи данных, отличным от заданного.
7. Следует игнорировать результаты запросов по гиперссылкам в тех случаях, когда ответ

сервера в http-заголовках не указан Content-type как text/html или Content-type отсутствует [10].
8. Страницы с одинаковым содержимым (дубли) считаются одной и той же страницей.
9. Ссылки должны проходить нормализацию [16].

Нормализация URL

Следуя [16], кратко опишем три основных типа нормализации, реализованные в RCCrawler.

I. Нормализация, сохраняющая исходное написание URL:
1. Конвертация протокола и хоста в нижний регистр. Пример:
http://eXamplE.com —> http://example.com.
2. Удаление порта по умолчанию. По умолчанию для доступа к сайту по протоколу http используется 80 порт. В случае протокола https по умолчанию используется 443 порт. Пример:
https://example.com:443/q.html —>
https://example.com/q.html.
3. Перевод закодированных знаком процента последовательностей в верхний регистр.
Пример:
http://example.com/%3a%2B –>
http://example.com/%3A%2B.

II. Нормализация, частично сохраняющая исходное написание URL:
1. Добавление завершающего слеша. Пример:
http://example.com/folder —>
http://example.com/folder/.
На практике данная процедура не всегда дает эквивалентный URL.
2. Нормализация пути. Символы перехода на уровень выше ("..") или текущего каталога (".") можно удалить до отправки http запроса по алгоритму, описанному в [16], раздел 5.2.4 Remove Dot Segments. Пример:
http://example.com/f1/./f2/../f3 —>
http://example.com/f1/f3.

III. Нормализация, полностью изменяющая исходное написание URL:
1. Отбрасывание идентификации фрагмента. Пример:
http://example.com/folder/#fragment —>
http://example.com/folder.
2. Замена двойных слешей на одинарный.
Пример:
http://example.com/f1//f2 —>
http://example.com/f1/f2.
3. Замена IP-адреса именем домена. Если мы знаем, что данному IP соответствует определенный домен, мы можем сделать замену IP на домен. Пример:
https:// 208.76.188.166 —>
https://example.com.
4. Сортировка параметров GET запроса.
Пример:
http://example.com/?b=1&a=3 —>
http://example.com/?a=3&b=1.
5. Удаление «?» при пустом списке параметров GET запроса. Пример:
http://example.com/? —> http://example.com/.
6. Удаление пустых параметров GET запроса.
Пример:
http://example.com/?a&b=1&c=2 —>
http://example.com/?b=1&c=2.

Общая архитектура RCCrawler

RCCrawler является многопоточным мультиплатформенным краулером, написанным на языке C++ стандарта C++11 с использованием фреймворка Qt версии 5.5 [14]. Он обладает общей архитектурой, которая может конкретизироваться определенными классами и, таким образом, реализовывать разные задачи, связанные с краулингом. На рисунке 1 представлена схема архитектуры краулера.

При запуске главного потока происходит чтение файла начальных установок *settings.ini* и выбирается реализация интерфейса **ApplicationManager**. Данный объект осуществляет управление приложением, инициализирует все другие потоки и ключевые объекты и завершает работу приложения. Например, потомок **ApplicationManager** может считывать при запуске необходимые данные из текстовых файлов или выполнять поступающие команды.

DownloadingThread представляет собой базовый класс для потока, занимающегося загрузкой страниц.

ParsingThread – базовый класс, потомки которого занимаются разбором скачанных страниц. Тут может быть любой разбор, не только извлечение ссылок, как в данной работе. Одновременно может быть несколько таких потоков, так как разбор страниц в данном случае – основная нагрузка.

RoutineThread – потомки этого класса должны заниматься протоколированием, переносом части данных в базу данных для уменьшения потребляемой оперативной памяти и другими операциями.

В многопоточном приложении необходимо согласовывать доступ к общим данным и минимизировать общение потоков. **DataManager** представляет интерфейс для получения потоками необходимых для работы данных. Потомки должны реализовывать блокировки для исключения так называемых «гонок за данными», когда несколько потоков модифицируют содержимое одной и той же области памяти.

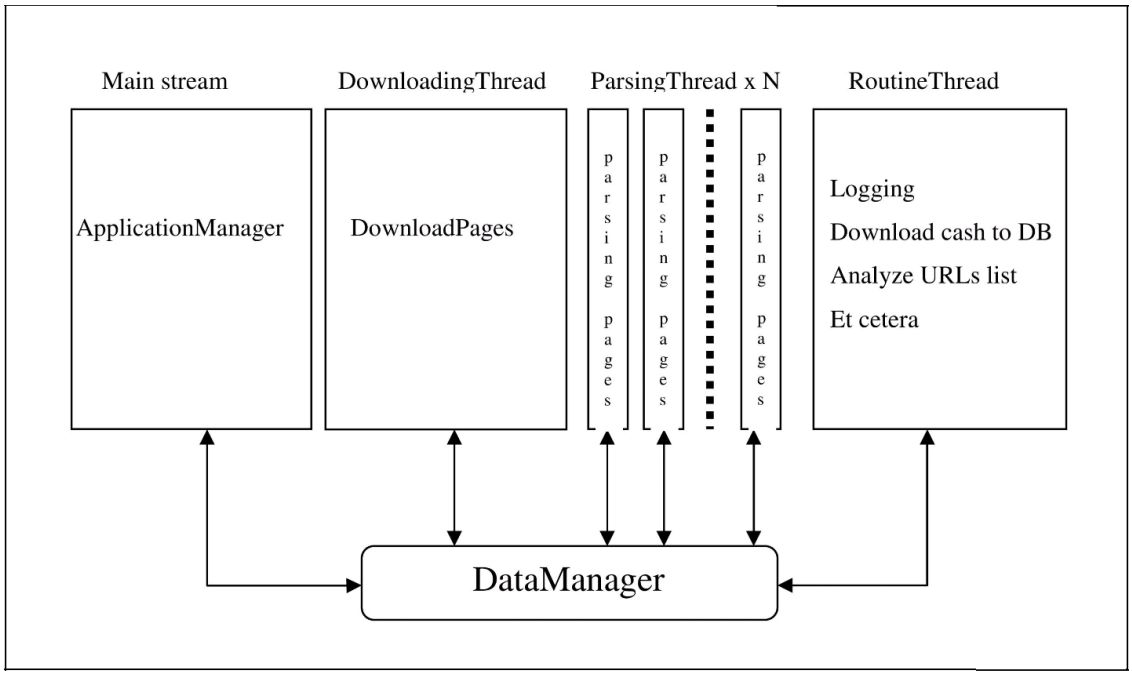

Рис. 1. Архитектура RCCrawler

DataManager делает общение потоков не столь необходимым. Также этот интерфейс дает возможность инкапсуляции данных, то есть совместного хранения данных и кода для их обработки полностью в оперативной памяти, во внешних источниках либо частично в оперативной памяти. Этот аспект касается только реализации **DataManager** и не влияет на другие компоненты RCCrawler.

За страницы сайта и связи между ними отвечают структуры **HostData**, **PageData** и **PageArc**. Рассмотрим их назначение и структуру.

HostData отвечает за сайт.

Поля:

QString host;
QString protocol;
QString str;
uint crawlDelay;
uint maxDownloadsAtTime;
int maxCrawlLevel;
RTRContainer rules;

здесь

host – строка с именем сайта (пример: "example.com");
protocol – протокол (пример: "https");
str – строка для быстрой подстановки (пример: "https://example.com");
crawlDelay – задержка по скачиванию в милисекундах (целое);
maxDownloadsAtTime – количество одновременных загрузок с сайта;
maxCrawlLevel – максимальный уровень краулинга (включительно);
rules – правила из robots.txt сайта.

PageData отвечает за страницу.

Поля:

ulong id;
HostData phD;*
QString url;
QString normalizedUrl;
ulong idFrom;
uint level;
uint outDegree;
bool blocked;
bool downloaded;
bool parsed;
QString content;
uint contentHash;
uint errorCode;
ulong replaceId;
uint downloadAttempts;
bool remove;

здесь

id – уникальный идентификатор, не должен быть равен нулю;
phD – указатель на соответствующую структуру HostData;
url – исходная ссылка;
normalizedUrl – нормализованная ссылка;
idFrom – id страницы, откуда получена ссылка на данную страницу;
level – уровень страницы;
outDegree – количество внешних исходящих ссылок;
blocked – этот параметр должен устанавли-

ваться ‘true’ в случае, если данная страница отправлена на скачивание, и вновь устанавливаться ‘false’ при обновлении;
downloaded – код, показывающий, была ли страница скачана;
parsed – код, показывающий, был ли совершен разбор html страницы;
content – содержимое страницы;
contentHash – хеш-таблица содержимого страницы. Если страница была разобрана, то ее содержимое хранить не нужно, а по анализу хеш-таблицы можно выявить, является ли страница дублем;
errorCode – код ошибки, 0 – если ошибки нет;
replaceId – id страницы, дублем которой является текущая;
downloadAttempts – количество попыток скачать страницу. Ограничение на этот параметр помогает избежать циклических редиректов;
remove – при обновлении PageData в хранилище DataManager сигнализирует о необходимости удаления и совершения сопутствующих действий.
PageArc отображает связь между двумя страницами.
Поля:
ulong id;
ulong from;
ulong to;
id – уникальный идентификатор, не должен быть равен нулю;
from – id PageData, на которой находится ссылка;
to – id PageData, на которую указывает ссылка.

Рассмотренные выше структуры являются основным набором данных, которые хранит DataManager и которыми оперирует все приложение. Их достаточно для построения вебграфа сайта.

Важными вспомогательными базовыми классами являются **RobotsTxt**, **ApplicationFinisher**, **ResultUnloader**.

RobotsTxt реализует получение файла robots.txt и проверку на следование правилам этого файла в процессе краулинга.

ApplicationFinisher выполняет проверку, нужно ли завершить приложение с использованием метода needToFinishApplication. Возможные варианты: вся работа выполнена, истек таймаут.

ResultUnloader осуществляет выгрузку результата в требуемом виде.

Конфигурация RCCrawler для решения поставленной задачи

В качестве **ApplicationManager** используется TextFileAM. Алгоритм его работы:
1. Создать и настроить реализации DataManager, DownloadingThread, заданное количество ParsingThread, RoutineThread, требуемые реализации ApplicationFinisher согласно файлу settings.ini (пример файла будет описан ниже).
2. Прочитать файл hosts.txt в качестве источника данных о сайтах для сканирования. Данный файл содержит построчно адреса сайтов и дополнительную информацию в следующем формате для каждой строки: protocol://host;maxDownloadsAtTime; maxCrawlLevel;crawlDelay.
3. Для каждого сайта получить правила из robots.txt и поместить стартовую страницу в DataManager.
4. Запустить потоки, определенные в п. 1.
5. Проверять в цикле с помощью реализаций ApplicationFinisher необходимость завершить приложение.
6. В случае положительного ответа от одного из ApplicationFinisher произвести выгрузку результата с помощью реализаций ResultUnloader, указанных в settings.ini.
7. Завершить приложение.

DownloadingThread в используемой конфигурации представлен классом **DT0**, который асинхронно загружает страницы, используя QNetworkAccessManager и сигнально-слотовую систему Qt. Упрощенный цикл работы:
1. Обработать события: принять сигналы и вызвать связанные слоты.
2. Запросить у DataManager коллекцию PageData для скачивания по каждому сайту.
3. По каждому сайту поставить данные на загрузку, если есть свободные соединения.
4. При выполнении требуемых условий отправить загруженные данные на обновление в DataManager.
ParsingThread представлен классом **PT0**. Упрощенный цикл работы: 1. Получить коллекцию PageData для разбора у DataManager.
2. Для каждой PageData:
- поставить в поле parsed ‘true’,
- разобрать content на ссылки.
3. По каждой ссылке:
- проверить, является ли она внутренней,
- если нет, то повысить outDegree; если да, то нормализовать ссылку и добавить в выходную коллекцию PDAndPACreateData (вспомогательная структура для обработки внутри DataManager и создания соответствующих данных).
4. Отправить выходную коллекцию PDAndPACreateData на вставку в DataManager.

5. Отправить входную коллекцию PageData в DataManager на обновление.

RoutineThread представлен классом **RT0**. Его единственной функцией является логирование (протоколирование).

Доступны три реализации ApplicationFinisher: **WorkIsDoneAF**, **TimeoutAF** и **StopFileAF**. Первая возвращает 'true' в методе needToFinishApplication, если у DataManager нет PageData для скачивания и обработки, вторая – по истечении таймаута. Так как прием пользовательского ввода во время работы невозможен по причине активного вывода информации по скачанным страницам и ошибкам, в качестве средства внешнего завершения приложения была разработана реализация StopFileAF, которая проверяет появление файла с заданным именем в каталоге приложения и сигнализирует о необходимости остановки в случае его наличия.

GephiSiteGraphRU является используемой реализацией ResultUnloader. Выгружает данные в формате для построения графа в Gephi – графической оболочке для представления и изучения графов [9].

В качестве DataManager используется **BMICDM** (Boost Multi-Index Container Data Manager). Для хранения структур PageData и PageArc используется Boost Multi-index Container Library [6]. Multi-index container представляет собой таблицу базы данных в памяти, доступ в которой осуществляется по индексам по полю или набору полей, и эти индексы задаются на этапе компиляции. Таким образом, доступ и поиск данных в такой коллекции обладает высокой производительностью. Также индекс может быть уникальным, и поэтому вставка с нарушением уникальности не будет происходить. Все эти качества multi-index container позволяют потокам находиться в ожидании разблокировки ресурса меньшее время.

Для экспериментов по проверке работоспособности RCCrawler использовался следующий файл *settings.ini*:

ApplicationManager = "TextFileAM"
ApplicationFinishers = "WorkIsDoneAF, TimeoutAF,StopFileAF"
DataManager = "BMICDM"
DownloadingThread = "DT0"
ParsingThread = "PT0"
ParsingThreadsCount = 2
RoutineThread = "RT0"
UrlAnalyzingThread = "EmptyUAT"
ResultUnloaders = "GephiSiteGraphRU,TestRU"
DownloadingThreadPDChunkSizeMultiplier = 5
ParsingThreadPDChunkSize = 20
DownloadingThreadSleepTime = 1
ParsingThreadSleepTime = 1
RoutineThreadSleepTime = 1000
CommonLogFile = "logs/common_log.csv"
ErrorLogFile = "logs/error_log.csv"
DisplayCommonLog = 1
DisplayErrorLog = 1
RobotsTxtClass = "OnlyDisallowRobotsTxt"
TimeoutSeconds = 240000
ResultFolder = "result"
SaveDuplicates = 1
StopFile = "stop.txt"

Эксперименты с RCCrawler

Проверка работоспособности RCCrawler была проведена, в частности, на сайтах двух тематических множеств: веб-сайты факультетов и институтов Санкт-Петербургского государственного университета (22 сайта) и веб-сайты институтов Карельского научного центра РАН (8 сайтов).

Результаты сканирования показывают очень большое разнообразие веб-сайтов как по количеству страниц (от 400 до 30 000), так и по количеству внутренних гиперссылок (от 14 000 до 1 570 000). Значение коэффициента ранговой корреляции Спирмена между этими характеристиками, равное –0.05, показывает слабую обратную корреляцию.

Неожиданными выглядят и некоторые другие результаты, например, среднее число ссылок, исходящих со страницы (по всем сайтам), равное 67.

Результаты сканирования частично можно было сравнить с результатами, полученными с помощью двух программ:
Screaming Frog [18] и ComparseR [8].

Screaming Frog – программа, разработанная в Великобритании, позволяющая в бесплатном режиме сканировать до 500 страниц/документов сайта, ComparseR – российская разработка с ограничением по сканированию в 150 html-страниц.

В обоих случаях получаем список html-страниц (начиная с начальной страницы), выдаваемых последовательно с возрастанием их уровня. К сожалению, список всех внутренних гиперссылок, связывающих эти страницы, ни в Screaming Frog, ни в ComparseR не создается. Для каждой найденной страницы указывается только URL страницы-предшественницы.

Достаточно условно можно выделить четыре следующие группы сайтов так, как это изображено на рисунке 2.

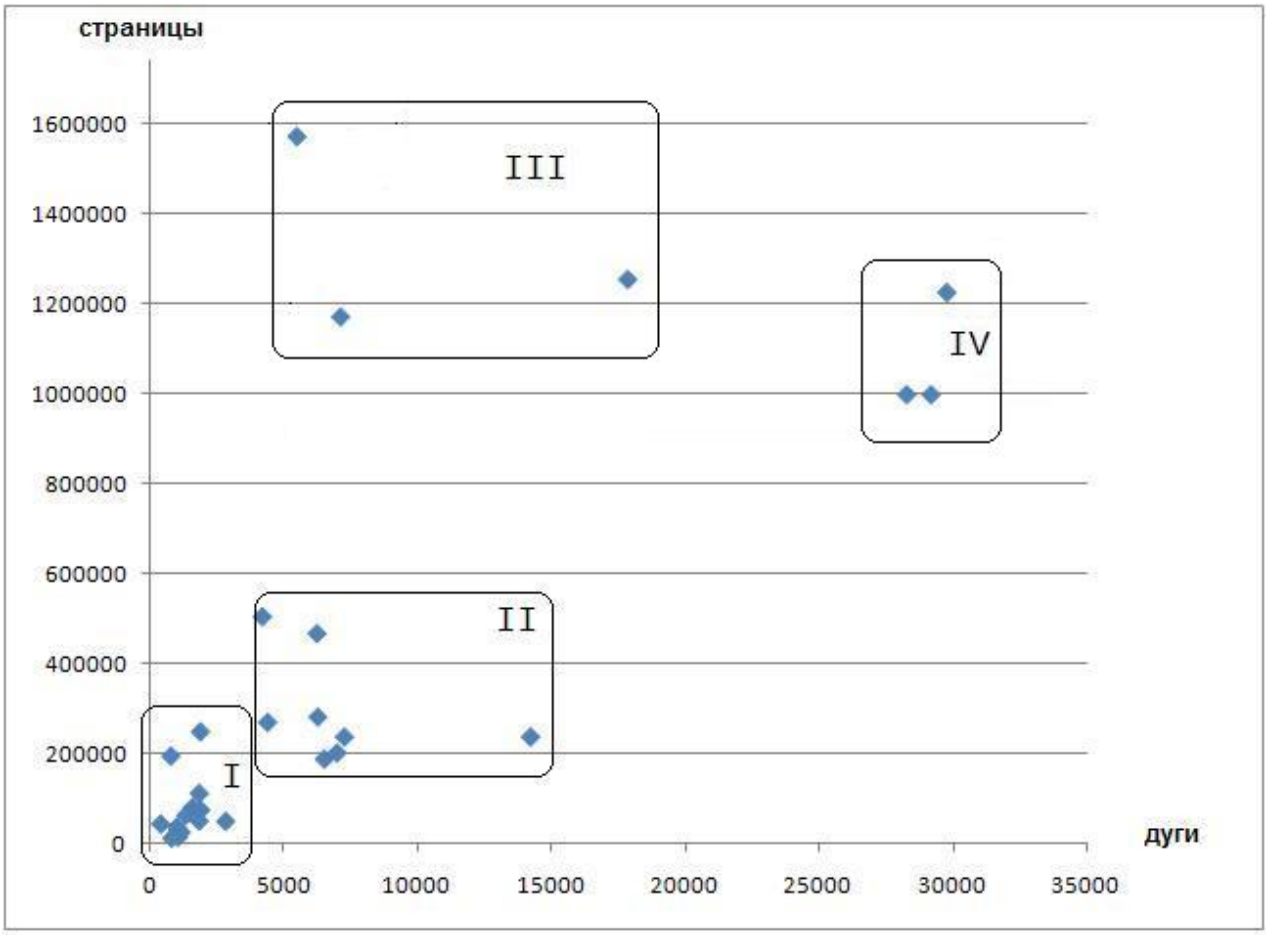

Рис. 2. Четыре группы сайтов

Основные результаты сравнения результатов краулинга кратко можно описать так.

Для группы I («маленькие» сайты с небольшим количеством страниц и ссылок) краулер RCCrawler выдает практически одинаковые результаты с ComparseR для первых 150 страниц и с Screaming Frog для 500 страниц.

Для группы II при тех же количествах сравниваемых страниц несовпадение результатов RCCrawler и ComparseR составляет до 15 %, а RCCrawler и Screaming Frog – до 20 %.

Для группы III несовпадение результатов RCCrawler и ComparseR составляет до 20 %, а для группы IV – до 25 %. Screaming Frog для сайтов групп III и IV генерирует до 25 % страниц с одинаковым контентом, поэтому мы его вынужденно исключили из сравнения в этих двух случаях.

Заключение

В работе приводится подробное описание основных требований, общей архитектуры и конфигурации краулера RCCrawler, предназначенного для решения достаточно узкой, но важной для исследований Веба задачи, а именно – сбора информации о веб-сайтах для последующего построения и исследования их веб-графов.

Одной из важных проблем, решаемых в RCCrawler, является проблема повторяющегося контента при разных URL. Эксперименты, проведенные на множестве университетских и научных веб-сайтов, показали хорошие результаты для относительно небольших сайтов и вполне удовлетворительные – для сайтов, содержащих до полутора десятков тысяч страниц и шестисот тысяч ссылок.

В научном плане разработанный RCCrawler позволит собрать экспериментальную базу для исследования таких важных вебометрических задач, как рациональная структура веб-сайта, глубина сканирования, сравнительный анализ структур различных сайтов, модели «серфинга» и др.

В практическом плане необходимо продолжить исследования результатов краулинга и реализации дополнительных возможностей RCCrawler, улучшающих результаты ра-

боты на очень больших сайтах. Реализация таких возможностей предусмотрена расширяемой архитектурой краулера.

Работа выполнена при частичной поддержке гранта РФФИ №15-01-06105 А «Разработка вебометрических и эргономических моделей и методов анализа эффективности присутствия в Вебе информационных вебпространств крупных организаций».

Литература

1. *Гиперссылка* // Википедия [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Гиперссылка (дата обращения: 15.04.2016).

2. *Горбунов А. Л.* Марковские модели посещаемости веб-сайтов // Интернет-математика 2007: сб. работ участников конкурса научных проектов по информационному поиску. 2007. С. 65–73.

3. *Левитин А. В.* Алгоритмы. Введение в разработку и анализ. М.: Вильямс, 2006. 576 с.

4. *Печников А. А., Чернобровкин Д. И.* Об исследованиях веб-графа сайта // Управление в технических, эргатических, организационных и сетевых системах: материалы конф. 2012. С. 1069–1072.

5. *An Extended Standard* for Robot Exclusion [Электронный ресурс]. URL: http://www.conman.org/people/spc/robots2.html (дата обращения: 16.04.2016).

6. *Boost* Multi-index Containers Library [Электронный ресурс]. URL: http://www.boost.org/doc/libs/1_58_0/libs/multi_index/doc (дата обращения: 17.04.2016).

7. *Baeza-Yates R., Castillo C.* Crawling the Infinite Web: Five Levels are Enough // Lecture Notes in Computer Science. Algorithms and Models for the Web-Graph, Third International Workshop. 2004. Vol. 3243. P. 156–167.

8. *ComparseR* – специализированная программа [Электронный ресурс]. URL: http://parser.alaev.info (дата обращения: 15.04.2016).

9. *Gephi* – The Open Graph Viz Platform [Электронный ресурс]. URL: https://gephi.org (дата обращения: 14.04.2016).

10. *HTTP/1.1:* Header field definitions [Электронный ресурс]. URL: https://www.w3.org/Protocols/rfc2616/rfc2616–sec14.html (дата обращения: 17.04.2016).

11. *HTTP 300* Status Codes [Электронный ресурс]. URL: http://developer.att.com/application-resource-optimizer/ docs/best–practices/ http–300–status—codes (дата обращения: 25.04.2016).

12. *Liu Y., Ma Z. M., Zhou C.* Web Markov Skeleton Processes and Their Applications // Tohoku Mathematical Journal. 2011. No. 63. P. 665–695.

13. *Pant G., Srinivasan P., Menczer F.* Crawling the Web // In Web Dynamics / M. Levene and A. Poulovassilis, eds. Springer, 2004. P. 153–178.

14. *Qt* – Home [Электронный ресурс]. URL: http://www.qt.io (дата обращения: 15.04.2016).

15. *RFC 1738* – Uniform Resource Locators. 1994. [Электронный ресурс]. URL: https://tools.ietf.org/html/rfc1738 (дата обращения: 25.04.2016).

16. *RFC 3986* – Uniform Resource Identifier. 2005. [Электронный ресурс]. URL: https://tools.ietf.org/html/rfc3986 (дата обращения: 16.04.2016).

17. *Schonfeld U., Bar-Yossef Z., Keidar I.* Do not crawl in the dust: different URLs with similar text // ACM Transactions on the Web. 2009. Vol. 3, no. 1. P. 111-–131.

18. *Screaming Frog* [Электронный ресурс]. URL: https://www.screamingfrog.co.uk/seo–spider (дата обращения: 25.04.2016).

19. *Status* codes in HTTP [Электронный ресурс]. URL: https://www.w3.org/Protocols/HTTP/HTRESP.html (дата обращения: 15.04.2016).

20. *Web crawler.* Open-source crawlers [Электронный ресурс]. URL: https://en.wikipedia.org/wiki/Web_crawler#Open-source_crawlers (дата обращения: 14.04.2016).

Поступила в редакцию 10.05.2016

References

1. *Hyperlink.* Wikipedia. URL: https://en.wikipedia.org/wiki/Hyperlink (accessed: 15.04.2016).

2. *Gorbunov A. L.* Markovskie modeli poseschaemosti saitov [Markov models of website visitation]. Internet-matematika 2007: sb. rabot uchastnikov konkursa nauchnyh proektov po informacionnomu poisku [Internet mathematics 2007: Proceedings of the contest of scientific projects for information retrieval]. 2007. P. 65–73.

3. *Levitin A. V.* Algoritmy. Vvedenie v razrabotku i analiz [Algorithms. Introduction to the design and analysis]. Moscow: Vil'ams, 2006. 576 p.

4. *Pechnikov A. A., Chernobrovkin D. I.* Ob issledovanijah web-grafa saita [On the research of site web-graph]. Upravlenie v tehnicheskih, ergaticheskih, organizacionnyh i setevyh sistemah: materialy konferencii [Control in technical, ergatic, organizational and network systems: Conference proceedings]. 2012. P. 1069–1072.

5. *An Extended* Standard for Robot Exclusion. URL: http://www.conman.org/people/spc/robots2.html (accessed: 16.04.2016).

6. *Boost* Multi-index Containers Library. URL: http://www.boost.org/doc/libs/1_58_0/libs/multi_index/doc (accessed: 17.04.2016).

7. *Baeza-Yates R., Castillo C.* Crawling the Infinite Web: Five Levels are Enough. Lecture Notes in Computer Science. Algorithms and Models for the Web-Graph, Third International Workshop. 2004. Vol. 3243. P. 156–167.

8. *ComparseR* – specialized software. URL: http://parser.alaev.info (accessed: 15.04.2016).

9. *Gephi* – The Open Graph Viz Platform. URL: https://gephi.org (accessed: 14.04.2016).

10. *HTTP/1.1:* Header field definitions. URL: https://www.w3.org/Protocols/rfc2616/rfc2616–sec14.html (accessed: 17.04.2016).

11. *HTTP 300* Status Codes. URL: http://developer. att.com/application-resource-optimizer/ docs/best–practices/http–300–status—codes (accessed: 25.04.2016).

12. *Liu Y., Ma Z. M., Zhou C.* Web Markov Skeleton Processes and Their Applications. Tohoku Mathematical Journal. 2011. No. 63. P. 665–695.

13. *Pant G., Srinivasan P., Menczer F.* Crawling the Web. In Web Dynamics. M. Levene and A. Poulovassilis, eds. Springer, 2004. P. 153–178.

14. *Qt – Home.* URL: http://www.qt.io (accessed: 15.04.2016).

15. *RFC 1738* – Uniform Resource Locators. 1994. URL: https://tools.ietf.org/html/rfc1738 (accessed: 25.04.2016).

16. *RFC 3986* – Uniform Resource Identifier. 2005. URL: https://tools.ietf.org/html/rfc3986 (accessed: 16.04.2016).

17. *Schonfeld U., Bar-Yossef Z., Keidar I.* Do not crawl in the dust: different URLs with similar text. ACM Transactions on the Web. 2009. Vol. 3, no. 1. P. 111–131.

18. *Screaming Frog.* URL: https://www.screamingfrog.co.uk/seo–spider (accessed: 25.04.2016).

19. *Status codes in HTTP.* URL: https://www.w3.org/Protocols/ HTTP/HTRESP.html (accessed: 15.04.2016).

20. *Web crawler. Open-source crawlers.* URL: https://en.wikipedia.org/wiki/Web_crawler#Open-source_crawlers (accessed: 14.04.2016).

Received May 10, 2016

СВЕДЕНИЯ ОБ АВТОРАХ:

Печников Андрей Анатольевич
главный научный сотрудник, к. ф.-м. н., д. т. н., доцент
Институт прикладных математических исследований Карельского научного центра РАН
ул. Пушкинская, 11, Петрозаводск,
Республика Карелия, Россия, 185910
эл. почта: pechnikov@krc.karelia.ru
тел.: (8142) 763370

Ланкин Александр Валерьевич
студент
Санкт-Петербургский государственный университет, факультет прикладной математики – процессов управления
Университетский пр., 35, Санкт-Петербург, Петергоф, Россия, 198504
эл. почта: edwvee@gmail.com
тел.: (812) 4287159

CONTRIBUTORS:

Pechnikov, Andrey
Institute of Applied Mathematical Research,
Karelian Research Centre, Russian Academy of Sciences
11 Pushkinskaya St., 185910 Petrozavodsk,
Karelia, Russia
e-mail: pechnikov@krc.karelia.ru
tel.: (8142) 763370

Lankin, Alexandr
Saint-Petersburg State University
Faculty of applied mathematics and control processes
35 Universitetskii prosp.,198504 Petergof,
Saint Petersburg, Russia,
e-mail: edwvee@gmail.com
tel.: (812) 4287159