электронный научно-технический журнал
ИНЖЕНЕРНЫЙ ВЕСТНИК
Издатель ФГБОУ ВПО "МГТУ им. Н.Э. Баумана". Эл No. ФС77-51036. ISSN 2307-0595

Метод автоматического обнаружения ошибок верстки веб-страниц

 # 05, май 2014
УДК: 004.91

Филиппов М. В., Белоус В. В., Бургина А. М.

Россия, МГТУ им. Баумана
filippov.mike@mail.ru
walentina.belous@gmail.com
anna190390@mail.ru

1. Введение

В последние годы сеть Интернет стремительно развивается. На сегодняшний день количество зарегистрированных в сети доменов составляет порядка 148 миллионов[1] и это число растет с каждым днем. Также растет и число пользователей интернета. По сравнению с данными 2000 года, это число выросло в 6,7 раз[2]. Создание собственного сайта для большинства компаний является необходимым фактором для успешного ведения бизнеса, так как позволяет значительно расширить круг клиентов. Вот почему конкуренция в этой сфере довольно высока, и даже незначительные ошибки при разработке сайта могут повлиять на выбор пользователя и способствовать его уходу к конкурентам. Поэтому необходимо ответственно подходить к разработке веб-приложений и уделять достаточно внимания каждому этапу разработки, в том числе этапу тестирования.

Когда речь идет о больших проектах, то ручного тестирования становится недостаточно, чтобы покрыть все возможные случаи и проверить каждую страницу, поскольку для этого необходимо очень много времени. В связи с этим используются различные средства для автоматизации процесса тестирования. Создание этих средств также требует немалых затрат, но в долгосрочной перспективе их использование продуктивнее, чем тестирование вручную.

В настоящее время существует немало средств автоматизированного тестирования сайтов. Самым популярным средством можно назвать валидатор W3C[3], который обнаруживает, например, следующие ошибки:

- отсутствие атрибута alt у картинок, чтобы в том случае, когда картинка не прогрузилась, был выведен текст,
- наличие незакрытых или неоткрытых тегов,
- наличие атрибутов, которые не поддерживаются данным типом документа.

Имеются также и другие средства (такие как Sitebeam[4], Wave[5]), которые наряду с вышеописанными функциями, позволяют обнаруживать ошибки javascript и css-стилей, определять, насколько сайт оптимизирован для поисковых систем.

Однако, прохождение всех этих проверок не гарантирует того, что веб-страница имеет хорошую верстку. Например, такие ошибки, как слияние цвета текста с цветом фона или выезд элемента-потомка за границы элемента-родителя, с помощью вышеуказанных средств обнаружены не будут.

Для того же, чтобы определить правильность верстки в конечном виде, необходимо проанализировать внешний вид страницы в конкретном браузере при определенном разрешении, потому как разные браузеры могут по-разному отображать одну и ту же страницу. И даже в пределах одного браузера при разных разрешениях могут быть отличия. На рисунке 1 можно видеть, как уменьшение разрешения приводит к некорректному отображению страницы.

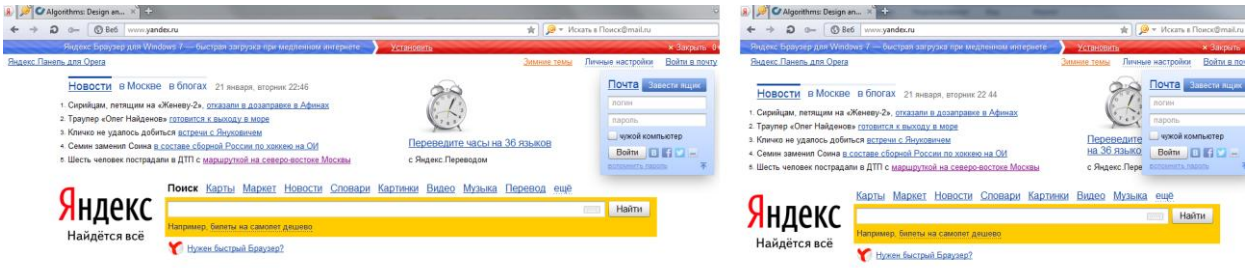

Рис 1. Наползание блока почты на блок рекламы при уменьшении разрешения.

Согласно [6], можно выделить 5 наиболее популярных браузеров (IE, Chrome, Opera, Firefox, Safari), суммарный процент использования которых превышает 95%. Также существует 9 наиболее распространенных разрешений экрана. При таком изобилии различных конфигураций необходимо средство, автоматизирующее процесс анализа внешнего вида страницы.

Задача автоматизированного тестирования верстки сводится к двум подзадачам – предоставление инструмента для проверки внешнего вида в нескольких браузерах и разрешениях и выработка правил поиска ошибок. Существующие решения (такие, например, как BrowserShots[7]) в основном занимаются первой подзадачей. Они позволяют пользователю выбрать из огромного набора интересующие его браузеры, загружают в них страницу и отображают получившиеся скриншоты, предоставляя пользователю самостоятельно анализировать их и искать ошибки.

Цель данной работы – создать инструмент, который будет решать обе подзадачи – не только загружать страницу в разных конфигурациях, но и автоматически находить ошибки.

В работе будет рассмотрен один из самых часто встречаемых типов ошибок верстки – неразличимость цветов текста и фона. В зависимости от того, насколько сильно они похожи, это может привести к тому, что пользователю либо придется вглядываться в текст, либо он просто не догадается о существовании, например, белого текста на белом фоне.

Соответственно, цель работы – создать метод, который будет выявлять ошибки такого типа.

Для достижения данной цели были решены следующие задачи:

- Разработан алгоритм определения цвета фона и цвета текста.
- Определение метрики, отражающие степень критичности ошибки.

2. Определение цветов текста и фона

Чтобы оценить степень различимости цвета текста и цвета фона, надо сначала их определить. В работе рассмотрен ряд подходов к решению данной задачи.

2.1 Однородный фон

Для начала будем считать, что фон задается одним цветом. В этом случае можно предложить следующие алгоритмы оценки степени различимости.

2.1.1 Получение вычисленных цветов

Самый простой и быстрый подход – использовать для каждого элемента вычисленные css-свойства color(цвет текста) и background-color (цвет фона). К сожалению, вычисленные свойства, возвращенные браузером, не учитывают прозрачность элементов. Соответственно, если у нас есть белый полностью прозрачный элемент, родитель которого имеет черный фон, то по алгоритму вместо верного черного цвета, мы получим белый, что будет являться ошибкой и приведет к неправильным вычислениям.

2.1.2 Расчет цветов с учетом прозрачности

Имея в виду недостаток первого подхода, необходимо учитывать прозрачность элементов. Прозрачность может быть указана в двух местах. Первое – свойство opacity, которое определяет уровень прозрачности элемента веб-страницы. В качестве значения определяется число из диапазона [0.0;1.0]. Значение 0 соответствует полной прозрачности элемента, а 1, наоборот - его непрозрачности. Второе возможное место указания степени прозрачности – непосредственно значения color и background-color, которые для прозрачных и полупрозрачных элементов будут заданы не в формате (r,g,b), а в формате (r,g,b,a), где a – степень прозрачности.

Таким образом, для того, чтобы получить результирующий цвет фона элемента, необходимо идти по всему дереву элементов сверху вниз и для каждого элемента вычислять цвет фона по следующей формуле:

$$background_{result} = a * background_{child} + (1-a) * background_{parent},$$

где $background_{child}$ – считанный цвет фона текущего элемента,

$background_{parent}$ – вычисленный цвет фона родителя,

a – степень прозрачности текущего элемента.

Цвет текста будет вычисляться похожим образом:

$$color_{result} = a * result_{child} + (1 - a) * background_{parent}$$

При использовании этого подхода корректный результат будет определяться всегда, когда речь идет о тексте на однородном фоне.

2.2 Фон с изображением

На многих сайтах вместо однотонного фона используется картинка. И хотя существуют рекомендации, предписывающие устанавливать цвет фона, который будет совпадать с цветом картинки (на случай отключенной загрузки картинок у пользователя), полагаться на это нельзя. Необходимо разработать алгоритм, который получал бы цвет картинки, чтобы принять его за цвет фона, и в дальнейшем сравнивать с цветом текста. Для этого было рассмотрено несколько подходов.

2.2.1 Получение картинки по адресу

Первый способ определения цвета картинки заключается в получении свойства элемента background-url (адреса картинки), загрузки ее и определения цвета. Этот способ имеет несколько существенных недостатков:

- background-url может быть указан по-разному: как закодированным в base64 бинарным файлом, так и адресом картинки, расположенной на удаленном сервере. Во втором случае, надо еще отдельно загружать картинку, что занимает немало времени.

- Существует техника css-спрайтов, когда несколько изображений располагаются в одном графическом файле. Для того, чтобы вырезать из файла нужный кусок, применяется CSS. Спрайты экономят трафик и ускоряют загрузку, поэтому часто используются на сайтах. Но задача определения цвета в случае таких изображений становится очень трудоемкой.

В связи с указанными проблемами данный подход использоваться не будет.

2.2.2 Получение цвета фона по одному скриншоту

Данный метод заключается в том, чтобы на скриншоте брать цвет левого верхнего пикселя элемента, полагая, что текста там нет. Достоинство данного метода – его простота, но он не работает для довольно часто встречающихся на страницах ссылок, где в левом верхнем углу оказывался именно текст.

2.2.3 Получение цвета фона по двум скриншотам

Для более точного отделения пикселей текста и фона предыдущий метод был усовершенствован путем использования двух скриншотов на основании следующего алгоритма:

Шаг 1.Сделать скриншот страницы.

Шаг 2.Поменять цвет каждой надписи на странице.

Шаг 3.Сделать второй скриншот.

Шаг 4.Найти внутренний пиксель элемента, у которого совпадает цвет на обоих скриншотах. Это и будет цветом фона.

Представленный выше алгоритм позволяет корректно обрабатывать случаи, которые неправильно решались при предыдущем методе – когда левый верхний пиксель занят текстом. Но метод будет ошибаться, если на фоне будут резкие смены контрастности. В этом случае, используя данный алгоритм, можно будет корректно найти некоторый пиксель фона, но его цвет не будет характеризовать основной цвет фона, на котором располагается текст. В данной работе эти случаи рассматриваться не будут.

В разрабатываемом методе было решено остановиться на данном алгоритме, потому что он дает приемлемую точность для большинства страниц.

3. Определение степени различия фона и текста(метрики)

После того, как цвет фона и цвет текста получены, надо выбрать метрику, которая будет определять степень различимости цвета текста и цвета фона. Будем считать, что у нас получились цвета в RGB-формате, где каждая компонента принимает значения из диапазона 0..255.

Рассмотрим несколько возможных вариантов метрики.

3.1 Нормированное расстояние в RGB-кубе.

Пусть есть два цвета (r_1, g_1, b_1) и (r_2, g_2, b_2). Тогда различием[8] этих цветов назовем величину

$$difference = \frac{\sqrt{(r_1-r_2)^2+(g_1-g_2)^2+(b_1-b_2)^2}}{255*\sqrt{3}},$$

где 255-нормировочный коэффициент, а $\sqrt{3}$ - длина диагонали куба.

3.2. Метрика W3C

В работе [9] предложен другой алгоритм для определения степени различимости цветов. Согласно нему, надо вычислить не только меру различимости цветов, но также и разницу в яркости. Яркость вычисляется по следующей формуле:

$$brightness = \frac{(r * 299) + (g * 587) + (b * 114)}{1000}$$

Для разности цветов используется следующее выражение:

$$difference = \big(max(r_1, r_2) - min(r_1, r_2)\big) + \big(max(g_1, g_2) - min(g_1, g_2)\big) + \big(max(b_1, b_2) - min(b_1, b_2)\big)$$

Согласно требованиям W3C, разница между яркостями должна быть больше 125, а разница между цветами – больше 500.

Если проанализировать данную формулу разности цветов, то получим, что максимальное значение разности будет (255-0) + (255-0)+(255-0) = 765. Исходя из этого выходит, что для цвета фона, сумма компонентов которого находится в интервале (264,500) не существует такого цвета текста, который бы дал нужную для различимости разницу.

3.3 Метрика HSP

Также было рассмотрено определение яркости по цветовой модели HSP [10].

$$brightness = \sqrt{0.299 * r^2 + 0.587 * g^2 + 0.114 * b^2}$$

3.4 Исследование метрик

Каждая метрика позволяет определить значение для пары цветов, но не указывает, насколько эта пара подходит для использования на странице. Таким образом, граничное значение, которое бы отделяло "хорошие" пары от "плохих", необходимо определить самостоятельно. В связи с тем, что граница W3C отбраковывает довольно значительное множество цветов, эксперимент по определению оптимального граничного значения должен быть проделан для нее наряду с другими метриками.

Для оценки каждой из представленных метрик был проведен следующий эксперимент. Случайным образом выбрано 100 пар цветов фона и текста и выведено на экран в виде текстового блока с предопределененной фразой. Далее эксперту было дано задание оценить каждый из текстов как «видимый хорошо» или «видимый плохо».

Для каждой метрики определения разности цветов была найдена оптимальная граница (граница, при которой процент неверных срабатываний минимален) и процент неверных срабатываний на этой границе. Результаты эксперимента представлены в Табл. 1.

Таблица 1. Результаты сравнения работы метрик определения разности цветов

Название алгоритма	Граница	Процент неверных определений
Нормированное расстояние в RGB-кубе	0,8	27,5
W3C разность цветов	174	26,3
W3C разность яркостей	31	22,8
HSP разность яркостей	48	15,7

Как можно видеть из таблицы, наилучший результат дала метрика HSP, которую и было решено использовать для определения разности цветов фона и текста.

На основании проведенных выше исследований можно описать представленный метод в виде следующих шагов:

- определение цвета текста и цвета фона по двум скриншотам (раздел 2.2.3);
- определение степени различимости цвета текста и цвета фона на основе метрики HSP.

4. Экспериментальное исследование метода

Для исследования качества работы метода было выбрано 20 веб-страниц. Каждая из них была проанализирована экспертами, а также программой на предмет наличия плохоразличимого текста. Результаты работы представлены в Табл. 2.

Таблица 2. Результат работы метода определения текста, который плохоразличим на фоне.

Номер страницы	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Ошибок найдено	3	5	0	1	3	2	5	0	2	2	1	0	0	1	1	1	0	3	0	1
Число найденных верно	3	2	0	0	3	2	4	0	1	2	1	0	0	1	1	1	0	2	0	1
Всего ошибок на странице	3	3	0	0	3	2	5	0	1	2	1	0	1	1	2	1	0	2	0	1
Количество пропусков	0	1	0	0	0	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0
Кол-во ложных срабатываний	0	3	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0

Из результатов эксперимента видно, что в большинстве случаев(65%) программа верно определяет все ошибки на странице. В остальных она либо находит лишнее, либо не находит имеющиеся, что связано с особенностью выбранного алгоритма определения цвета фона. Но стоит отметить, что таких неверных срабатываний на каждой странице совсем мало(1-2).

Результат верной работы разработанного метода можно видеть на рис. 2 и рис. 3, где красными рамочками выделены найденные программой тексты, которые слишком близки по цвету с фоном.

.

Рис. 2. Найденные совпадения цветов на сайте http://kroshka-peggi.svinki.ru/

Дата выпуска	Номинал	Наименование	Сплав	Монетный двор	Тираж (млн. экз.)	Цена (руб.)
4.05.2000	2 рубля	Ленинград	Медно-никелевый	СПМД	10	30-50
4.05.2000	2 рубля	Москва	Медно-никелевый	ММД	10	30-50
4.05.2000	2 рубля	Мурманск	Медно-никелевый	ММД	10	30-50
4.05.2000	2 рубля	Новороссийск	Медно-никелевый	СПМД	10	30-50
4.05.2000	2 рубля	Смоленск	Медно-никелевый	ММД	10	30-50
4.05.2000	2 рубля	Сталинград	Медно-никелевый	СПМД	10	30-50
4.05.2000	2 рубля	Тула	Медно-никелевый	ММД	10	30-50

Рис. 3. Найденные совпадения цветов на сайте http://realcoin.ru/

Как было сказано выше, программа может давать неверный результат, когда фоновое изображение имеет переменную контрастность. Пример такого ложного срабатывания можно видеть на рис. 4 , где программа определила цвет фона по верхним светлым пикселям и решила, что они слишком близки по цвету с белым текстом, хотя на самом деле текст находится на более темном фоне и никакой ошибки там нет

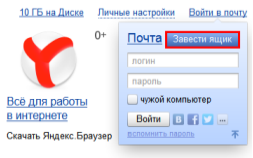

Рис. 4. Ложное срабатывание на сайте http://www.yandex.ru

5. Заключение

В данной работе представлен метод определения ошибок верстки типа «неразличимый текст». Для этого был разработан алгоритм для получения цвета фона и цвета текста элемента, а также исследованы имеющиеся метрики для определения меры различия цветов и выбрана оптимальная.

Далее было произведено исследование эффективности предложенного метода и оно показало хороший результат: в 65% случаев метод не дал ни ложных срабатываний, ни пропусков.

Для уменьшения количества ложных срабатываний в дальнейшем можно усовершенствовать алгоритм выбора внутренних пикселей для определения цвета фона. Для этого необходимо поменять порядок обхода пикселей и начинать не с левого верхнего угла, как это делается в настоящий момент, а с пикселей, находящихся ближе к тексту. Также возможно использование нескольких пикселей и дальнейшее их «усреднение».

Список литературы

1. Domain Counts Internet Statistics. — http://www.whois.sc/internet-statistics/. — 2013
2. World internet usage and population statistics — http://www.internetworldstats.com/stats.htm. — 2012
3. Markup Validation Service. — http://validator.w3.org/ —2013
4. Sitebeam —http://sitebeam.net/-2013
5. Wave —http://wave.webaim.org/-2013
6. Global stats —http://gs.statcounter.com/. — 2013
7. Browsershots - http://browsershots.org/-2014
8. D.H.Alman Industrial color difference evaluation.- Color Research and Application, No. 18 (1993), pp. 137-139
9. C.Ridpath, W.Chisholm. Techniques For Accessibility Evaluation And Repair Tools. - http://www.w3.org/TR/AERT#color-contrast/ - 2000
10. R.F.Darel HSP Color Model — Alternative to HSV (HSB) and HSL. - http://alienryderflex.com/hsp.html/ - 2006