

University of Louisville

ThinkIR: The University of Louisville's Institutional Repository

Electronic Theses and Dissertations

12-2008

Focused image search in the social Web.

Zhiyong Zhang

University of Louisville

Follow this and additional works at: <https://ir.library.louisville.edu/etd>

Recommended Citation

Zhang, Zhiyong, "Focused image search in the social Web." (2008). *Electronic Theses and Dissertations*. Paper 1639.

<https://doi.org/10.18297/etd/1639>

This Doctoral Dissertation is brought to you for free and open access by ThinkIR: The University of Louisville's Institutional Repository. It has been accepted for inclusion in Electronic Theses and Dissertations by an authorized administrator of ThinkIR: The University of Louisville's Institutional Repository. This title appears here courtesy of the author, who has retained all other copyrights. For more information, please contact thinkir@louisville.edu.

FOCUSED IMAGE SEARCH IN THE SOCIAL WEB

By

Zhiyong Zhang

Supervisor: Dr. Olfa Nasraoui

A Dissertation Submitted To the Faculty of the
Graduate School of the University of Louisville
in Fulfillment of the Requirements
for the Degree of

Doctor of Philosophy

Department of Computer Engineering and Computer Sciences
University of Louisville
Louisville, Kentucky

December 2008

FOCUSED IMAGE SEARCH IN THE SOCIAL WEB

By

Zhiyong Zhang
Supervisor: Dr. Olfa Nasraoui

A Dissertation Approved On

12/15/2008

Date

by the following Committee

Dissertation Director

ACKNOWLEDGEMENTS

I would like to thank my supervisor, Dr. Olfa Nasraoui, for her great effort in advising me on this project. Many times, she has been working around the clock to recommend additions and revisions to my work. I really appreciate her critical guidance and instructions. I would like to thank my wife, Michelle. Without her support, I could not have finished my doctorate study in three years. I also thank Dr. Frigui and my lab member, Carlos, with whom I worked when I started the very preliminary ideas in this work as a course project. Thanks to all the other lab members, Nurcan, Esin, and Sofiane, and all the friends who helped do the evaluations. At last, I would like to thank all the committee members, Dr. Adel, Elmaghraby, Dr. Ibrahim Imam, Dr. Ming, Ouyang, and Dr. Joanna Wolfe, for their constructive advice. Last but not least, I thank my external committee members Dr. Ricardo Baeza-Yates, and Dr. Roelof van Zwol, from Yahoo Research, who have gracefully contributed to this work with their feedback and encouragement.

ABSTRACT

FOCUSED IMAGE SEARCH IN THE SOCIAL WEB

Zhiyong Zhang

December 15th, 2008

Recently, social multimedia-sharing websites, which allow users to upload, annotate, and share online photo or video collections, have become increasingly popular. The user tags or annotations constitute the new multimedia “meta-data”. We present an image search system that exploits both image textual and visual information. First, we use focused crawling and DOM Tree based web data extraction methods to extract image textual features from social networking image collections. Second, we propose the concept of visual words to handle the image’s visual content for fast indexing and searching. We also develop several user friendly search options to allow users to query the index using words and image feature descriptions (visual words). The developed image search system tries to bridge the gap between the scalable *industrial image search engines*, which are based on *keyword* search, and the slower content based image retrieval systems developed mostly in the academic field and designed to search based on image content only. We have implemented a working prototype by crawling and indexing over 16,056 images from *flickr.com*, one of the most popular image sharing websites. Our experimental results on a working prototype confirm the efficiency and effectiveness of the methods, that we proposed.

TABLE OF CONTENTS

	Page
ACKNOWLEDGEMENTS	iii
ABSTRACT	iv
LIST OF TABLES	x
LIST OF FIGURES	xi

CHAPTER

I	Introduction	1
1	Keyword Based Image Search and CBIR	1
1.1	Problem with Keyword Based Image Search	1
1.2	Content Based Image Retrieval to the Rescue	2
1.3	Integrating Keyword Search and CBIR	3
2	Social Networking	3
3	Vertical Search and Focused Crawling	4
4	Contributions	4
II	Background and Literature Review	7
1	General Web Image Text Extraction Methods	7
2	Extracting Data from Structured Websites	8
3	Focused Crawling	10
4	Image Ranking	11
5	Image Pre-processing Tasks Relevant to Search	11
5.1	Image Anisotropic Diffusion	12
5.2	Image Segmentation	14
5.3	Image Gridding and Image Tiles	14
6	Indexing Schemes	15
6.1	GLA (K-means algorithm)	15

6.2	Trees, VA-files, and Inverted files	16
7	Existing Query By Image Content Systems	19
7.1	Query by Image Color	19
7.2	Query by Example Images	21
7.3	Relevance Feedback in QBE	21
7.4	Query Interface	22
7.5	Query Result Ranking	23
III Detail Record Page Data Extraction		24
1	Introduction	24
2	Structured Organization of Photo-sharing Sites	26
3	Problem Definition	27
4	DOM Tree Path String	29
4.1	Path String, Perceptual Group, and Semantic Group . . .	29
4.2	Path String Observations	31
4.3	Path String and Path String Node Value Pair	33
5	DRP Identification	33
5.1	Record Title Uniqueness and Cohesiveness	33
5.2	Record Feature Pairs	35
5.3	DRP page classifier	40
6	DRP Data Extraction Based on Path String	41
6.1	Structured Organization of DRPs	41
6.2	Path String based Data Extraction	41
6.3	Case study: applying DTPS methods on Flickr	44
7	Summary	47
IV Focused Image Crawling		48
1	Introduction	48
2	Motivation for Profile-based Focused Crawling	49
2.1	Popularity of Member Profile	49
2.2	Typical Structure of Social Media-sharing Websites . . .	50

2.3	Profile-based Focused-crawling	52
3	Path String based Page Classification	52
3.1	Classifying Pages based on Real Data Path Strings	52
4	Profile based Focused Crawler	54
4.1	Ranking from the Inner Profile	54
4.2	Ranking from the Inter Profile	55
4.3	Combining The Inner-Rank and Inter-Rank	56
5	Co-tagging Topic Discovery	57
6	Profile-based Focused Crawling System	58
6.1	Co-tagging Topic Discovery Stage	58
6.2	Profile-based Focused Crawling Stage	60
7	Summary and Discussions	63
V	Image Indexing	64
1	Introduction	64
2	Image Indexing	65
2.1	Sparseness of Image Features	65
2.2	GLA Clustering and Cluster-Merge Algorithm	66
2.3	Building an Inverted Index for Image Content with “Field Boosting”	69
3	Image Codebook Design and Feature Extraction	70
3.1	Image Color Codebook	71
3.2	Extracting and Indexing Image Color Features	72
3.3	Image Tiling for Flexible Querying and Focused Retrieval	74
3.4	Image Texture Codebook	77
3.5	Extracting and Indexing Texture Features	79
3.6	Training Set and Classifier Selection	81
4	Indexing Image Tags and Content	82
4.1	Image Index Boosting	82
4.2	Combined Indexing Scheme	82

5	Summary and Discussion	83
VI Image Querying		85
1	Introduction	85
2	Query By Image Tags	85
3	Query By Image Content	87
3.1	Query By Image Color	87
3.2	Query By Image Texture	88
4	Query Interface	89
5	Query Result Ranking	90
6	Query Expansion	91
7	Summary	92
8	Open Problems and Discussion	93
VII Experimental Results		94
1	Demo Development Stages	94
1.1	Stage 1: Show and Tell (text and color)	94
1.2	Stage 2: Text, Color, Texture	94
1.3	Stage 3: Final Prototype	95
2	DOM Tree Path String Data Extraction	95
2.1	DRP Classification Experiment	95
2.2	DRP Extraction Experiments	96
2.3	Path String Differentiability Experiments	98
3	Profile-based Focused Crawling	100
3.1	Topic Discovery Through Co-tagging	100
3.2	Profile Based Focused Crawling	101
4	Indexing Image Content	105
4.1	Effectiveness of Texture Word Representation and Comparison with CBIR	105
5	Current Working Prototype Evaluation	108
5.1	Deployment Configuration	108

5.2	Snapshot 1: Content Color vs Keyword Color	109
5.3	Snapshot 2: Texture Words Filter Irrelevant Images . . .	110
5.4	Snapshot 3: Texture Words Help Find Target Images . .	110
5.5	Snapshot 4: Use “minus” option to filter irrelevant images	111
6	Image Search Demo Evaluation	111
6.1	Search Functionality Evaluation of Second stage develop- ment	111
7	Search Functionality Evaluation of 3rd stage development . . .	113
7.1	Focused Crawling vs Breadth First Crawling (BFS) Com- parison	113
7.2	Search Functionality Evaluation	114
VIII Conclusions and Future Work		119
1	Conclusions	119
2	Future Work	121
REFERENCES		122
CURRICULUM VITAE		132

LIST OF TABLES

TABLE	Page
1 Comparison of Keyword and Content Based Search	3
2 Comparison of Query Functionalities of Image Search Systems	23
3 Query Functionality Comparison 2	89
4 DRP Classification Accuracy Result	96
5 Data Extraction Comparison	97
6 Path Strings Differentiability	99
7 top co-tagging tags for “flowers”	100
8 top co-tagging tags for “nyc”	101
9 COREL Categories of Images Tested	105
10 Demo Links	108
11 Users’ Evaluation of Search Functionality	112
12 Profile based Focused Crawling vs Breadth First Crawling	113
13 Precision and Recall Comparison	114
14 Crawling Topic	115
15 Search Comparison Between Tags and Tags Plus Contents	117

LIST OF FIGURES

FIGURE	Page
1 Google Advanced Image Search	1
2 Flickr and Google Results for Tiger	2
3 R-tree Structure	17
4 VA-file Structure	18
5 Inverted File Structure	18
6 fotolia	20
7 istockphoto	20
8 yotophoto	21
9 Flickr Image Textual Information	26
10 Springer	28
11 Dom Tree Representation	29
12 feature pair	36
13 example feature pairs	37
14 Flickr Image Page DOM tree	46
15 Extracting Textual Information from Flickr Image Page	46
16 List Page	50
17 Detail Page	50
18 Typical Structure of A List Page, Detail Pages, and Profile Pages . .	51
19 classifier	53
20 inner rank	55
21 co-tagging	58
22 co-tagging diagram	59
23 focused crawling stage	61
24 Log-log plots of different quantization method	66

25	Quantization Results Comparison	67
26	Indexing Image by Inverted File	69
27	Process of Generating Image Color Codebook	71
28	Example of a Color Codebook	71
29	Extracting Colorwords	73
30	Tile Selection	74
31	Texture Image Without Large Segmentation Blocks	75
32	Process of Generating Image Texture Codebook	78
33	Example Texture Codebook	79
34	Extracting Texturewords	80
35	Indexing Images	83
36	Current Query Interface	90
37	Crawling Harvest Ratio	102
38	Crawling Comparison	103
39	Crawling Comparison	104
40	Crawling Comparison	104
41	Texture Precision Results	106
42	Texture Number of Returned Results	107
43	Retrieval Efficiency	108
44	Red Rose Query	109
45	Query for Dog	110
46	Query for Florida Beach	110
47	Query for Disney	111
48	Evaluation Images	116
49	Evaluation Images	117

CHAPTER I

Introduction

1 Keyword Based Image Search and CBIR

1.1 Problem with Keyword Based Image Search

Current large scale image search engines like Google Image Search ¹, and Yahoo Image Search ², provide only keyword based search functionality.

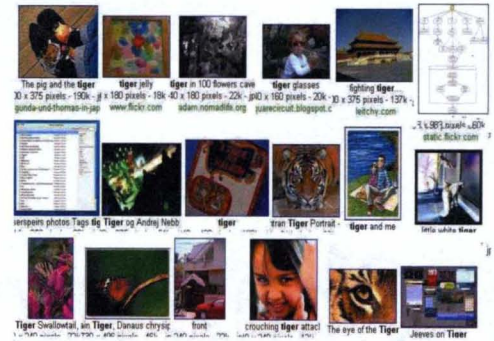
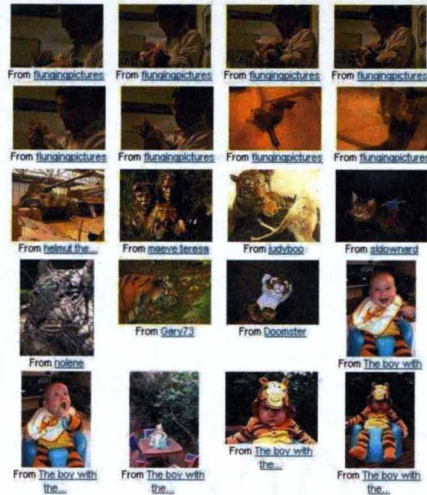
The image shows the Google Advanced Image Search interface. At the top, the Google logo is followed by "Advanced Image Search" in a blue bar. Below this, there's a section for "Find results" with four radio button options: "related to all of the words", "related to the exact phrase", "related to any of the words", and "not related to the words". Each option has a corresponding text input field. To the right of these fields is a "Google Search" button. Below the "Find results" section, there are several filter categories: "Size" (Return images that are) with a dropdown menu set to "any size"; "Filetypes" (Return only image files formatted as) with a dropdown menu set to "any filetype"; "Coloration" (Return only images in) with a dropdown menu set to "any colors"; "Domain" (Return images from the site or domain) with a text input field; and "SafeSearch" with three radio button options: "No filtering", "Use moderate filtering" (which is selected), and "Use strict filtering". At the bottom right, there is a copyright notice "©2006 Google".

Figure 1: Google Advanced Image Search Functionalities.

Figure 1 shows the search functionalities provided by Google advanced image search. For Coloration options, it has “any colors”, “black and white”, “grayscale”, and “full color”, options, which are not sufficient to differentiate between different images.

¹<http://images.google.com/>

²<http://images.search.yahoo.com/>



(a) Flickr Search Results (b) Google Search Results On Flickr

Figure 2: First page of returned results of the query “tiger”.

Figure 2 shows the search results of using Flickr’s image search and Google image search on Flickr. We can see that the search results contain too many concepts and the above search interfaces provide no functionality to filter through different concepts. Given that any popular search keyword will return tens of thousands of image search results, it then becomes the users’ burden to scan through several pages of image search results to find the right image. The user could also input more keywords to narrow the search. However, narrowing the search results by adding more keywords can result in reduced coverage because of the insufficiency of annotations.

1.2 Content Based Image Retrieval to the Rescue

Research in CBIR has illustrated the merits of exploiting low level image features for image retrieval. If we can provide a simple interface to help users sift through images by using image content, we may be able to lower the coverage lost while improving the precision at the same time. However, on *large scale Web repositories*, most CBIR systems' performance may become questionable from a scalability point of view. Moreover, they generally do not provide keyword based

search functionalities which may be useful in certain search scenarios.

1.3 Integrating Keyword Search and CBIR

TABLE 1
Comparison of Keyword and Content Based Search

Type	Framework	Weakness	Strength
Keyword Based	Search Engines	Information Expr- ession Difficulties	Well-Developed Text Retrieval Methods; Scalability; Easy to Use
Content Based	CBIR	Semantic Gap and Scalability	Ability to express the user's information need using content

As can be seen from Table 1, CBIR and commercial keyword based image search tend to complement each other. Thus, combining them together could tap on the strengths of a wealth of well-developed text retrieval techniques and on the robustness and scalability of current Search Engine technologies to handle image search without being blind to the wealth of information within an image's "content". A combined tool also promises to alleviate the weaknesses of either type of search working in isolation.

2 Social Networking

As social networking has recently begun to gain increasing popularity, photo sharing and video sharing sites such as Flickr, Youtube, etc, have started attracting more and more users. Generally, social networking sites contain certain conceptual social structure, such as user profiles, users' groups, users' uploaded photos, and users' comments. By integrating the collaborative annotations together with low level image features in a seamless way for searching, we may be able to present users with a powerful tool to control and manage these huge image collections.

3 Vertical Search and Focused Crawling

Vertical search engines have been proposed to handle specialized search demands. Comparing to the “all-in-one” style “whole-web” search, they can provide more accurate and customized results in specialized fields. Given a model of a few specific topics (for example, important keywords from a certain category), a focused web crawler can automatically acquire the web pages of specified topics and discard irrelevant topics when crawling. By indexing the crawling results acquired by the focused crawler, we can then build realize a vertical search engine on some specified topics. This combination not only provides users with more accurate specialized results, but also helps alleviate the scalability burden of crawling and indexing the entire world wide web.

4 Contributions

This proposal mainly focuses on focused crawling and searching of images with both textual tags and content information. For this purpose, we have implemented a preliminary prototype with promising results. The prototype can be accessed at <http://webmining.spd.louisville.edu:8080/isearch/>. In this research work, we have made the following contributions.

1. We propose the DOM Tree Path String (DTPS) web data extraction method. This extraction method is simple, intuitive and quite effective according to our preliminary tests. It can readily be expanded to structured or semi-structured web data extraction.
2. We design and build a focused crawler based on our proposed two-stage crawling scheme. We focus our crawling study on the photo-sharing site Flickr.com. For this site, we propose profile-based best-first-crawl and crawl-list expansion through image co-tagging. The proposed methods can be easily expanded to other photo-sharing or social networking sites.
3. We devise a framework for the seamless integration of keyword and content

based image search. This mixed search provides users with a powerful tool to search large image collections on the web. Compared with “keyword-only” based image search, we provide more functionalities by integrating “content-based” image retrieval techniques. Compared with traditional CBIR, we use more scalable and efficient indexing and searching techniques. We also present some analytical and empirical analysis and methods to ensure the optimality of the integrated inverted index.

4. We provide innovative indexing and ranking schemes both for keyword based image search and content based image search. More specifically, for keyword based image search, we integrate many “social information” aspects in photo-sharing sites such as tags, groups, and comments. For content based image search, we expanded the inverted file indexing scheme to handle image content and to differentiate the importance of different “spatial” regions within an image.
5. Based on image segmentation and image gridding, we can analyze and index an image based on the “spatial distribution” of its content. Segmenting the image into parts has additional benefits since it helps increase the sparseness of our color or texture thesaurus, making our inverted-file indexing structure more justified and scalable. In our case, we will only consider segmentation by image color, because texture features are too expensive to be included in the segmentation stage.
6. Our codebook design techniques for image color and texture rely on scalable data mining techniques, and also take into account the various scalability and quality “constraints” of text-based indexing and retrieval.

CHAPTER II

Background and Literature Review

In this chapter, we will review topics that are related to focused image crawling and content based image search. As we cover the methods and techniques that are currently being used, we will discuss general web image text extraction methods, focused crawling and web data extraction, image ranking schemes, relevant image processing techniques including diffusion, segmentation, and gridding; the indexing schemes including R-trees, VA-files, and inverted index; the Generalized Lloyd Algorithm for image codebook generation, and end with a review of current content based image search systems. These will form the basis for our further exploration.

1 General Web Image Text Extraction Methods

To extract textual annotations for images in the World Wide Web, several methods have been explored. Associated text such as image file names, captions, alternate text, hyperlinks, HTML titles, etc are analyzed to build an image search system [20]. The problem of how to associate this text with the images has also been explored. In [26], text from *one-step links* (surrounding text in the same page except image captions) and *two-step links* (in-link or out-link pages) were used for image indexing and retrieval. The normalized cumulative term weights for these textual documents are used for ranking the result images with respect to given keywords. In [55], the *weight ChainNet* method, which is based on *lexical chains* and is claimed to differentiate the importance of different texts by assigning different *lexical chain* names to image title, alt, caption and page title, was proposed and a *list space model* was later used for similarity matching. Later in [22], different

terms such as *TM* (page oriented text), *LT* (link oriented text), and *BT* (caption oriented text) were used to differentiate different types of associated texts and several heuristics were explored for term weighting.

2 Extracting Data from Structured Websites

Research about extracting data from structured web sites has always been important. As most web sites would use web page templates and databases to generate pages automatically, the problem becomes the template deduction. RoadRunner [13] [23], takes one HTML page as the initial wrapper, and uses *Union-Free Regular Expression* (UFRE) method and *Align, Collapse under Mismatch, and Extract* (ACME) matching techniques to generalize the wrapper under mismatch when parsing the other HTML pages with the wrapper. Several heuristics such as *Terminal-Tag Search, Recursion, Backtracking*, were used to handle mismatches for wrapper generalization. However, as pointed by [4], the union-free assumption of the “grammar” of the template is not valid for many collections of pages that do contain disjunctive items and the scalability concern of the search heuristics used for solving mismatches is also not well addressed in RoadRunner. The authors in [4] then developed the EXALG extracting system, which is mainly based on extracting LFEQs (*Large and Frequently occurring Equivalence classes*) and differentiating roles of tokens using *dtokens* to deduce the template and extract data values. The basic assumption of this method is that two tokens rarely occur in a large number of pages with the same frequency by chance. The “back-end” reason for this same frequency occurrence is that they are generated by the same template. Compared to RoadRunner, this method allows disjunctions. Later in [69], a tree similarity matching method was proposed to extract web data, where a tree edit distance method and a *Partial Tree Alignment* mechanism were used for aligning tags in the HTML tag tree. This was an improvement over the previous MDR [38] method by the same authors. This method is based on the observation that a data region usually contains a list of similar data records such as search engine results. But their method won’t be able to detect single occurrence

data records such as the title or the uploader of an image in image sharing sites.

Research in extracting web record data has widely used a web page’s structure [37] and a web page’s visual perception pattern. In [24], the authors proposed several filter rules to extract content based on the Document Object Model (DOM) tree. They developed a human interaction interface through which users were able to customize which type of DOM-nodes are to be filtered. While their target was for general HTML content and not for web record, neither did they suit their methods to structured data record extraction. In [72], [73], and [74], the authors proposed to use the tag tree structure and visual perception pattern to extract data from search engine results. They used several heuristics to model the visual display pattern that a search engine results page would usually look like, and combined this with tag path. Compared with their tag path method, our *Path String* approach keeps track of all the parent-child relationship of the DOM nodes in addition to keeping the parent-first-child-next-sibling pattern originally used in the DOM tree. We also include the node property in the *Path String* generation process. Another work that used visual perception information to extract data records is ViPER [57], which used region weighting and data alignment.

Wrapper generation and template detection are closely related to each other. In [75], the authors proposed a method for joint optimization of wrapper generation and template detections. They proposed to group pages together based on their DOM-tree structure similarity, and then to do template detection on these same types of pages. Our *DRP* (Detail Record Page) identification process shares the same motivation, while our method was able to cover pages from different web sites. The degree of human intervention has always been a factor to consider when developing a wrapper. In [30], the authors developed a semi-automatic wrapper to minimize human intervention, while in [29], the authors proposed several heuristics for title extraction from HTML documents. In [62], the authors presented “C-repeated pattern” methods for data extraction and label assignment. Attribute labeling was also addressed in [78].

3 Focused Crawling

Focused crawling was introduced by [10], in which three components, a classifier, a distiller, and a crawler, were combined to achieve focused crawling. They used a Bayes rule-based classifier [9], which was based on both texts and hyperlinks. The distillation process involves link analysis similar to *hub* and *authority* method. In [43], the authors presented a comparison of different crawling strategies such as breadth-first, best-first, pagerank, and shark-search. Further in [48], the authors presented a comparison of different classification schemes used in focused crawling and concluded that Naive Bayes was a weak choice when compared with Support Vector Machines or Neural Networks.

In [3] [2], the authors presented a probabilistic model for focused crawling based on the combination of several learning methods. These learning methods include content based learning, URL token based learning, link based learning, and sibling based learning. Their assumption is that pages which share similar topics tend to link each other. On the other side, authors of [15] [28] explored using context graph for building a focused crawling system. The two-layer context graph and Bayes Rule based probabilistic models were used in both systems.

Instead of using page content or link context, another work by Vidal et al [61] explored the page structure for focused crawling. This *structure-driven* method shares the same motivation with our work in trying to explore specific page-layouts or structure. In their work, each page was traversed twice: the first pass for generating the navigation pattern, and the second pass for actual crawling. In addition, some works [79] [51] for focused crawling used meta-search methods, that is, they based their method on taking advantage of current search engines. Among these two works, Zhuang et al [79] used search engine results to locate the home pages of an author and then used a focused crawler to acquire missing documents of the author. Qin et al [51] used the search results of several search engines to diversify the crawling seeds. and This puts the accuracy of their system at the mercy of other related search engines.

In [1], the authors used *cash* and *credit history* to simulate the page importance and implemented an *OPIC* (Online Page Importance Computation) strategy based on the web pages linking structure (*cash flow*).

4 Image Ranking

Current commercial *image* search engines tend to inherit the traditional *text* search ranking methods. Several object-level ranking schemes were introduced in recent years. In [47], *Poprank* was introduced. Several heuristics such as *authored-by*, *cited-by*, *published-by*, etc, were adopted to rank web objects such as books, commercial products, etc. In [70], Zhang et al presented methods to build a vertical image search engine for searching high quality photos. BM25 ranking and score fusion methods were used. Web spam poses serious challenge to normal ranking schemes. To combat spam, [68] introduced *Topical TrustRank*, which is an improvement over *TrustRank*, and [6] introduced *hilltop* experts-target ranking method.

5 Image Pre-processing Tasks Relevant to Search

For those working in the data mining field, data pre-processing is a very familiar term. Through the crucial KDD (Knowledge Discovery in Data) step of data pre-processing, we achieve the objective of removing noise and extracting useful features that can help improve the results of the data mining task. Image pre-processing shares some similar characteristics with data pre-processing, and can be used to obtain cleaner data for better feature extraction. Here, we will discuss several image processing techniques that are relevant to our work. These include image anisotropic diffusion, image segmentation, and image gridding.

5.1 Image Anisotropic Diffusion

In order to obtain a better image segmentation, it is desirable to first smooth the image to remove noise. However we don't want to blur the image so that we end

up missing the edge boundaries. In [49], Perona et al proposed *anisotropic diffusion*, which encourages intra-region smoothing while inhibiting inter-region smoothing. The basic idea is to apply heat diffusion concepts to an image. By setting the conduction coefficient to 1 in the interior of each region, and to 0 at the region boundaries, the blurring of images would then take place separately in each region with no interaction between regions. The *anisotropic diffusion* filtering process is given by the following transformation.

$$I_s^{t+1} = I_s^t + \frac{\lambda}{|\eta_s|} \sum_{p \in \eta_s} g(\nabla I_{s,p}) \nabla I_{s,p} \quad (1)$$

where I_s^t is a discretely sampled image, I_s^{t+1} is the diffusion result, s denotes the pixel position in a discrete, two-dimensional (2-D) grid, and t denotes discrete time steps (iterations). The constant $\lambda \in \mathbb{R}^+$ is a scalar that determines the rate of diffusion, η_s represents the spatial neighborhood of pixel s , and $|\eta_s|$ is the number of neighbors (usually four, except at the image boundaries). The image gradient (magnitude) in a particular direction is given by:

$$\nabla I_{s,p} = I_p - I_s^t, \quad p \in \eta_s. \quad (2)$$

while $g(\cdot)$ is the edge stopping function. By defining $\psi(x) = \rho'(x) = g(x)x$, Black et al [7] relate the *anisotropic diffusion* to the robust estimation problem in robust statistics. They formalize the problem to finding an image I that satisfies the optimization criterion:

$$\min_I \sum_{s \in I} \sum_{p \in \eta_s} \rho(I_p - I_s, \sigma) \quad (3)$$

where $\rho(\cdot)$ is a robust error norm and σ is a “scale” parameter. They showed that an appropriate choice of the ρ -function can minimize the effect of the outliers, or $(I_p - I_s)$, at the boundaries between piecewise constant image regions. Equation 3 can be further solved by gradient descent

$$I_s^{t+1} = I_s^t + \frac{\lambda}{|\eta_s|} \sum_{p \in \eta_s} \psi(I_p - I_s^t, \sigma) \quad (4)$$

where $\psi(\cdot) = \rho'(\cdot)$. Equation (4) turns out to be the same as Equation (1). They then discussed several choices of the robust error norms or ρ -functions.

- *Lorentzian error norm:*

$$\rho(x, \sigma) = \log[1 + \frac{1}{2}(\frac{x}{\sigma})^2], \quad \psi(x, \sigma) = \frac{2x}{2\sigma^2 + x^2} \quad (5)$$

It has the same format as one of the functions discussed in [49].

- *Huber's minimax norm:*

$$\rho(x, \sigma) = \begin{cases} x^2/2\sigma + \sigma/2 & |x| \leq \sigma, \\ |x|, & |x| > \sigma \end{cases}, \quad \psi(x, \sigma) = \begin{cases} x/\sigma, & |x| \leq \sigma, \\ \text{sign}(x), & |x| > \sigma \end{cases} \quad (6)$$

- *Tukey's biweight norm:*

$$\rho(x, \sigma) = \begin{cases} \frac{x^2}{\sigma^2} - \frac{x^4}{\sigma^4} + \frac{x^6}{3\sigma^6} & |x| \leq \sigma, \\ \frac{1}{3}, & |x| > \sigma \end{cases}, \quad \psi(x, \sigma) = \begin{cases} x[1 - (x/\sigma)^2]^2 & |x| \leq \sigma, \\ 0, & |x| > \sigma \end{cases} \quad (7)$$

By comparing these three norms, Black et al [7] found out that Tukey norm produced sharper boundaries. The Tukey biweight norm was adopted in [50] for image retrieval. For these reasons, we will adopt the *Tukey biweight norm* for our image *anisotropic diffusion* process.

5.2 Image Segmentation

It is typically desired to segment an image into several parts: a background part, and one (or several) object part(s). This corresponds to our human's visual perception process: when we look at our surroundings, our visual system will segment the object(s) from the background subconsciously, and even without our knowledge, while in the image processing field, segmenting an image remains a quite challenging task and is still an open and active research field.

One significant breakthrough in image segmentation is the *Normalized Cut* or *Ncut* [56] method. In [56], Shi et al used spectral graph theory to partition the image into different segments. While Shi et al provided sound theoretical background for their minimum normalized cut method, they didn't give a solution for computing the minimum normalized cut effectively and efficiently for practical

use. Later in [17], Pedro et al developed an efficient method for graph based segmentation, which uses a Kruskal’s algorithm style method for greedily merging different components (image regions), thus having the computation complexity of $O(n\log(n))$ for n image pixels. In our work, we will use the spectral graph method mentioned above given in [56] and [17].

5.3 Image Gridding and Image Tiles

Dividing an image into small grids or sub-blocks is another way of partitioning an image. In [14] [52], etc. Dagli et al, and Rahman et al all divide the image into 4×4 sub-blocks or 16×16 sub-grids for further processing and analysis. Dividing the image into grids not only implements a coarse image segmentation, but also helps users to extract Regions of Interest (ROI)s of the image.

6 Indexing Schemes

In this section, we will first review an algorithm that is closely related to our indexing method: the GLA (Generalized LLoyd Algorithm) or K-means algorithm, then we will study several indexing schemes including R-trees, VA-files, and inverted index methods.

6.1 GLA (K-means algorithm)

In order to build an inverted index, we need to transform the image content features into textual words. One way to achieve this goal is to use some Vector Quantization (VQ) methods. A simple and intuitive way of quantization method is to use a Uniform Quantizer, as was already implemented in a prior version of our image indexing and search system [71]. Since there are several advantages of using Nonuniform Quantization such as an improved Signal to Noise Ratio (SNR) (see [21]), we will try the Nonuniform Quantization method. Among the Nonuniform Quantization methods, The Generalized Lloyd Algorithm (GLA), also known as k-means algorithm in clustering domain or LBG algorithm in the data compression

literature, is widely used for generating image codebooks for indexing or retrieval. See [41] [58] [32] and [44].

The GLA algorithm, as described in [21], is given below.

Step 1. Begin with an initial codebook C_1 . set $m = 1$.

Step 2. Given a codebook, $C_m = \{y_i; i = 1, \dots, N\}$, find the optimal partition into quantization cells, that is, use the Nearest Neighbor Condition to form the nearest neighbor cells:

$$R_i = \{x : d(x, y_i) < d(x, y_j); \text{all } j \neq i\}. \quad (8)$$

If x yields a tie for distortion, e.g., if $d(x, y_i) = d(x, y_j)$ for one or more $j \neq i$, then assign x to the set R_j for which j is smallest.

Step 3. Using the Centroid Condition, find $C_{m+1} = \{\text{cent}(R_i); i = 1, \dots, N\}$, the optimal reproduction alphabet (codebook) for the cells just found.

Step 4. Compute the average distortion for C_{m+1} . If it has changed by a small enough amount since the last iteration, then stop. Otherwise set $m + 1 \rightarrow m$ and go to Step 2

In [40], Ma et al give a clustering-like method by sequentially increasing the number of colors to cluster the colors in that region until either (a): the number of colors has reached the maximum number of colors allowed; or (b): the mean squared error of the color clustering is below a pre-defined threshold. This approach will guarantee that the number of colors in the codebook extracted from any image is optimally small and further guarantee the sparseness of the image colors in the codebook, but the computation cost may be too expensive. In this method, the mapping relation is not fixed, which means that the same pixel value may be mapped to one codeword for one image while it may be mapped to another codeword for another image.

6.2 Trees, VA-files, and Inverted files

QBIC [19], Blobworld [8], WALRUS [46], and WBIIS [64], which uses the QBIC system, all use *R*-trees* [5] for indexing. The *R*-tree* is a variant of the *R-tree* by Guttman [25]. Besides *R*-tree*, there are other tree-based methods like *SR-tree* [36] and *M-tree* [11] etc. All claim to have good performance for nearest neighbor or similarity search. This is understandable because these tree-based methods tend to organize the data into similar partitions or clusters. Thus the nearest neighbor search can avoid traversing many branches of the tree, and so it is efficient. Figure 3 shows the basic *R-tree* indexing structure.

The *curse of dimensionality* spells potential trouble for all the above tree-based methods. In [65], Weber et al showed that the performance of these tree-based methods degrades significantly as the number of dimensions increases and is even outperformed by a sequential scan whenever the dimensionality is above 10. Weber et al then proposed a scheme named *vector approximation file* (or ‘*VA-file*’) and demonstrated that *VA-file* can offer better performance for high dimensional similarity search. An *OVA-file* (*Ordered VA-file*) structure, which places *approximations* that are close to each other in close positions for later fast access, was later used in [39] for video retrieval. Figure 4 gives the basic *VA-file* indexing structure.

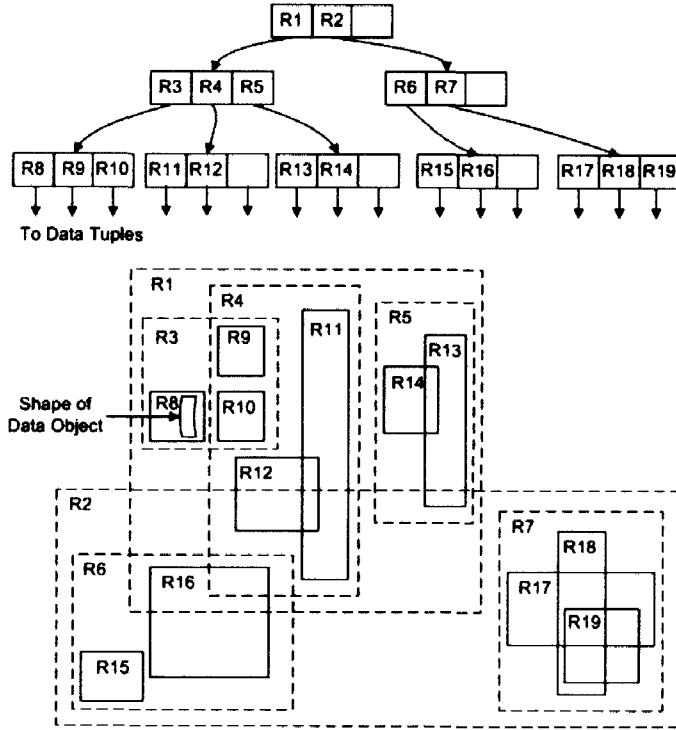


Figure 3: R tree indexing structure.

The *VA-file* structure adopts a signature-file like filtering method, thus trying to build a mechanism for fast scanning for nearest neighbors search. Other signature-file based indexing systems include work by Essam et al [16].

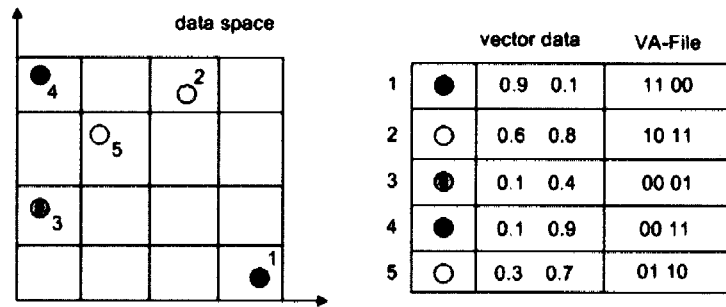


Figure 4: VA-File indexing structure.

However, in [67] [80], Zobel et al showed that signature files are distinctly inferior to inverted files as data structure methods for indexing. Although their

performance evaluations were mainly based on Boolean queries, their test data were for text indexing, and they didn't address the similarity query or the nearest neighbor query, the success of inverted files on textual search engines has recently started attracting the attention of researchers. Figure 5 shows an inverted file indexing structure.

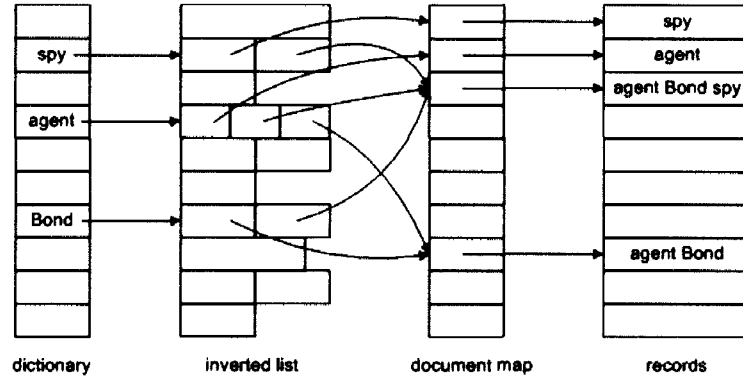


Figure 5: Inverted File Indexing Structure.

Although there is no corroborative evidence showing that the inverted-file methods should be expanded to multimedia retrieval area, scalability concerns have drawn attention to the potential of using inverted files for multimedia indexing. The Viper [59] system had attempted to use inverted files for indexing image databases and their experimental results showed that their inverted indexing scheme had better performance than the vector space system used before by the same authors. In [33], Jing et al tried to use a modified inverted file for image indexing and showed through experiments that inverted file indexing is much more efficient than sequential search without much loss in accuracy. However, their modification is more akin to a query expansion style in the query phase rather than a substantial modification in the indexing phase. In [52], Rahman et al also used inverted files for image indexing and observed comparable accuracy results for inverted file indexing and sequential search.

7 Existing Query By Image Content Systems

7.1 Query by Image Color

Querying by image color has been gaining popularity both in the academic and commercial domains. More and more systems claim to have or plan to have the *query-by-image-color* functionality. In the academic field, a few Web prototypes and small scale museum art work search portals allow users to search by image color. Retrievr ¹, based on [31], allows users to query by *sketch* and by *image*. For the sketch query, users can select the color of the sketch from a color palette, which contains 12 colors in the first level and 72 different grades for each color in the second level. The QBIC system has also found its application in Museum Digital collection search ², where users can search by choosing colors from a palette and assigning color proportions. Another search system for fine art ³ also allows search-by-color, although the color query can not be combined with other types of queries. In the commercial field, some websites, especially some stock photo sites, have begun to integrate color search to gain more profit. Fotolia ⁴, iStockphoto ⁵, and yotophoto ⁶ all have search-by-color functionality. Comparing with works in the academic domain, these sites allow users to combine keywords and colors for search on collections that are of much larger scales than traditional CBIR. Thus their objectives and scale are inline with our proposed work.



Figure 6: fotolia.

¹<http://labs.systemone.at/retrievr/>

²<http://www.hermitage.ru/cgi-bin/db2www/qbicColor.mac/qbic?selLang=English>

³http://www.artsugar.com/search/search_page

⁴<http://www.fotolia.com/>

⁵<http://www.istockphoto.com/>

⁶<http://yotophoto.com/>

Advanced Search

Keywords:
(separate words with spaces, AND, OR, or NOT)

File type:

Member name:

Minimum size: wide by high in pixels

Search Tips

The Controlled Vocabulary organizes and standardizes the search terms used at iStockphoto.

Search by Color

Color:

Simple Web RGB HSV

CopySpace™

Use the 3x3 grid to choose the area you would like your copy space to appear. It's real, it works, and it's only on iStockphoto.

Sensitivity:

Figure 7: istockphoto.

Search for these words: Exclude these words:

☒ Match all words ☐ Match any words

Licenses: ☒ Creative Commons (CC) ☒ GNU FDL (GL) ☒ Public Domain (PD) ☒ Site specific (SS)

Size: Orientation:
 Any
 Horizontal/Landscape
 Vertical/Portrait
 Square

Color:
 Pick a color or enter hex value

help
 help desk
 search basics
 advanced search

about
 about this site
 about the images
 press

Figure 8: yotophoto.

7.2 Query by Example Images

Most current CBIR systems have QBE (*Query By Examples*) functionality, where the user uploads an “image” and expects results that are similar. This includes some CBIR systems in the academic field including QBIC, SIMPLIcity, CIRES, IKONA ⁷, Retrievr, MARS ⁸, etc. In the commercial area, Tiltomo ⁹ and Riya ¹⁰ also have image search by example functionality. Because query by example alone is barely sufficient for most search scenarios in real life, some researchers have tried to combine QBE (*Query By Example*) with QBK (*Query By Keyword*) together to obtain systems that combine keywords and visual features together for

⁷www-rocq.inria.fr/cgi-bin/imedia/ikona

⁸<http://www.ifp.uiuc.edu/qitian/MARS.html>

⁹<http://www.tiltomo.com/>

¹⁰<http://www.riya.com/>

described in section 5, and in the data extraction process described section 6.

2 Structured Organization of Photo-sharing Sites

As discussed in chapter II, the general methods for extracting web image text information struggle to differentiate different types of text by assigning corresponding weight to different types, and they miss one important issue of the whole problem: the noisy web environment. Here, noise refers to the fact that the surrounding text contains too much information which is irrelevant to the embedded images.

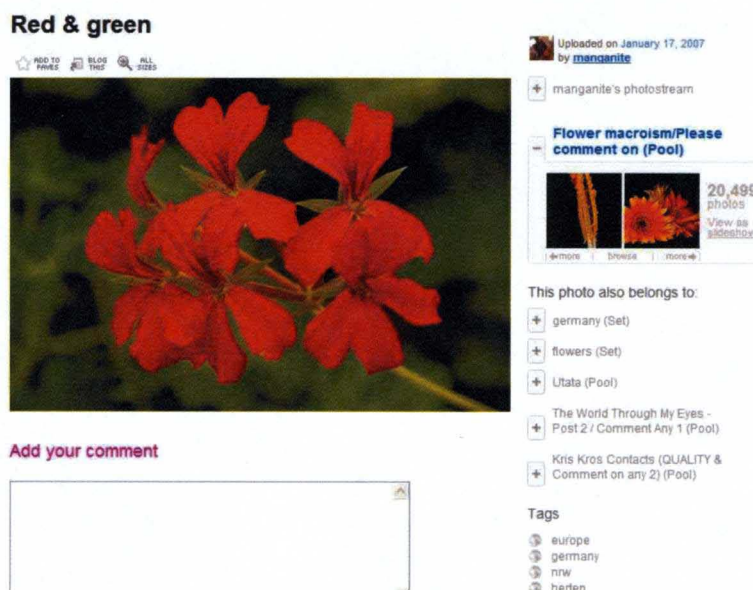


Figure 9: Flickr Image Textual Information.

It is very hard to extract accurate textual annotations when faced with significant noise in the web environment. Moreover, if we use similar methods to deal with images in photo-sharing sites, we would miss the important photo organizing structures of these sites. For example, if we take a closer look at a typical Flickr image page, as shown in Figure 9, it is not hard for us to discern certain patterns. For instance, the image title has a bigger font than image tags; image tags, sets and groups (pools) are clickable and aligned in parallel; etc.

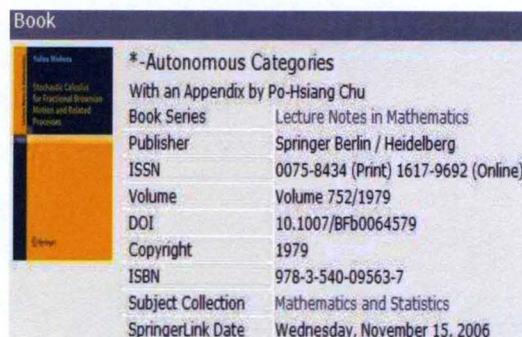
Similarly, in some other photo-sharing sites, we can find certain patterns. This page display pattern does not occur only in one page. Many other pages

(actually all the detailed image pages) look similar. This phenomenon is not accidental. It is due to the way by which these web pages are designed and maintained. All these pages are generated by one single template with different data being inserted in fixed positions, it would thus be strange if they don't show similar layout patterns. This give us an important hint to exploit to improve accuracy. The motivation is to deduce the hidden template from the web page layout. In the following, we will relate this problem to the problem of extracting data from structured or semi-structured websites and set this problem of structured web data extraction from photo-sharing sites to the more general problem of *DRP* (Detail Record Page) information extraction, which is defined and explained below.

3 Problem Definition

To make our problem clear, we will first give a definition of our object term: *DRP* (*Detail Record Page*).

Definition: A *DRP* (*Detail Record Page*) is a web page that contains only one record with detailed descriptions of the record.




Book	
	*-Autonomous Categories
	With an Appendix by Po-Hsiang Chu
	Book Series Lecture Notes in Mathematics
	Publisher Springer Berlin / Heidelberg
	ISSN 0075-8434 (Print) 1617-9692 (Online)
	Volume Volume 752/1979
	DOI 10.1007/BFb0064579
	Copyright 1979
	ISBN 978-3-540-09563-7
	Subject Collection Mathematics and Statistics
SpringerLink Date Wednesday, November 15, 2006	

Figure 10: Springer Book Page.

A *DRP* can be a book or an article page in a digital library, a product page in an e-commerce site, or an image or video page in a social media sharing site, etc. For example, Figure 10 give an example *DRP*.

Problem Statement: Our purpose for the data extraction is to: (1)

Identify *DRPs* in the Web pages automatically with minimum human intervention;
(2) Deduce web-site-dependent schemas for *DRPs* and use the schemas for accurate and efficient detail record data extraction.

The first step, *page identification*, of our work, paves the way for the second step, *schema generation*. Once the we have deduced the schema, we will use it to reinforce the page identification process, which further play a verification role of the page identification and hence the schema itself.

One naive way to extract *DRP* data from the web is to enumerate a list of popular websites and *manually* deduce a schema for each one. However, the fast growth of the web record pages would outpace any such manual schema deduction attempts. Also, the popularity of different web sites changes quickly and dramatically. To this end, an automatic page identification and schema deduction tool with minimum human intervention is needed.

4 DOM Tree Path String

To begin the discussion of our extraction methods, let's illustrate the basic term that we will use for both page identification and data extraction, *DTPS (DOM Tree Path String)*, which we will define soon.

4.1 Path String, Perceptual Group, and Semantic Group

The *DOM (Document Object Model)* defines a hierarchy of *Node* objects. Among the different types of nodes, *Element Node*, and *Text Node* are mostly used by us. Figure 11 give a figure 9's *DOM* tree representation.

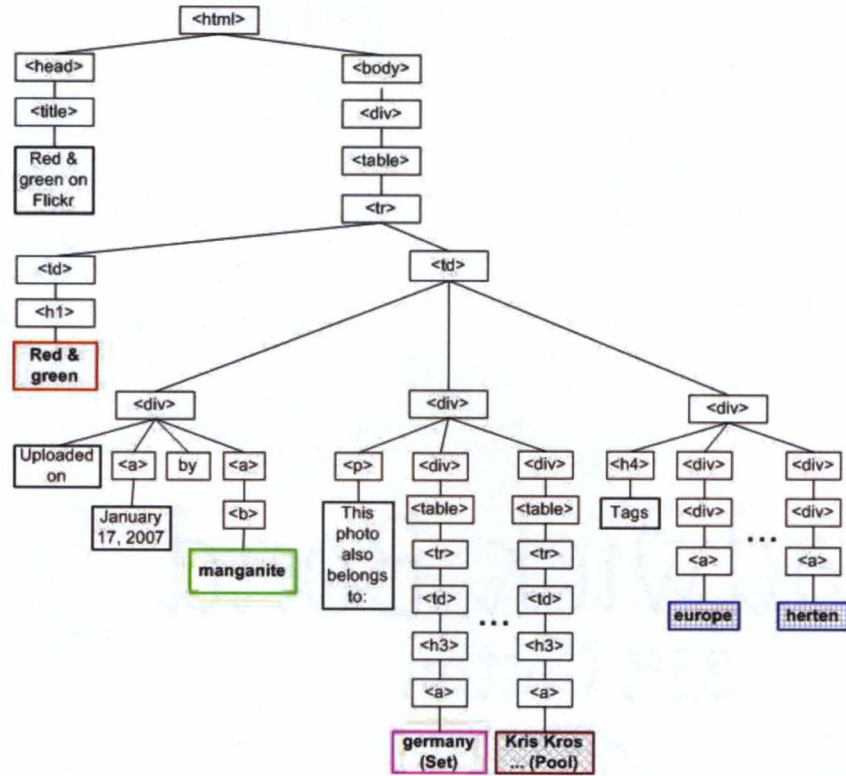


Figure 11: Dom Tree of Figure 9 (different color corresponds to different *Semantic Groups*).

In Figure 11, the whole DOM tree can be seen as a *Document Node*, whose child is the *Element Node* `<html>`, which further has two children `<head>` and `<body>`, both *Element Nodes*, etc. The *Element Nodes* are all marked with `<>` in Figure 11. At the bottom of the tree, there are a couple of *Text Nodes*. In the DOM Structure Model, the *Text Nodes* are not allowed to have children, so they are always the leaf nodes of the DOM tree. There are other types of nodes such as *CDATASection Nodes* and *Comment Nodes* that can be leaf nodes. *Element Nodes* can also be leaf nodes. *Element Nodes* may have properties. For example, “`<TDclass = “TagList” >`” is an *Element Node* “`<TD >`” with property “`class = “TagList”`”. Readers may refer to <http://www.w3.org/DOM/> for a detailed specification.

Definition 1: A *Path String* of a node is the string concatenation from the node’s immediate parent all the way to the tree root. If a node in the path has

properties, then all the *display* properties should also be included in the concatenation. We use “-” to concatenate a property name and “/” to concatenate a property value.

For example, in Figure 11, the *Path Strings* for “europe” and for “herten” are “<a×div×div×div-class/TagList><td-class/RHS><tr×...×body×html>”, and the *Path String* for “Red & Green” is “<h1×td×tr×...×body×html>”.

Note when we concatenate the property DOM node into *Path Strings*, we only concatenate the *display* property. A *display* property is a property that has an effect on the node’s outside display. Such properties include “font size”, “align”, “class”, etc. Some properties such as “href”, “src”, “id”, etc., are not *display* properties as they generally do not affect the appearance of the node. Thus, including them in the *Path String* will make the *Path String* over-specified. For this reason, we will not concatenate these properties in the *Path String* generation process.

Definition 2: A *Perceptual Group* of a web page is a group of text that look similar in the page layout. For example, “germany (Set)” and “Kris Kros ... (Pool)” are in the same *Perceptual Group* in Figure 9; and “europe” and “herten” are also in the same *Perceptual Group*.

Definition 3: A *Semantic Group* of a web page is a group of text that share the same category of meanings. For example, “europe” and “herten” are in the same *Semantic Group* as they all belong to the image’s tags, while “germany (Set)” and “Kris Kros ... (Pool)” are in different *Semantic Groups* as they belong to “Set” and “Pool” respectively.

4.2 Path String Observations

We propose to use the *Path String* for page identification and schema generation as it has the following benefits.

Path String Efficiency

First, when we extract *Path Strings* from the *DOM* tree, we save lot of space, since we do not need to save a *Path String* for every *Text Node*. For example, we

only need *one Path String* to represent *all* different “tags” in Figure 11, as all these “tags” share the same *Path String*. Second, transforming the tree structure into linear string representation will reduce the computation complexity. We will explain this more in our data extraction process.

Path String Differentiability

Now let’s have a closer examination of the DOM tree in Figure 11. Using our *Path String* definition, we know that *Text Nodes* “europe” and “herten” share the same *Path String*. Interestingly, they share a similar appearance when displayed to users as an HTML page, thus we say that they are in the same *Perceptual Group*. Moreover, their display property (*Perceptual Group*) is different from that of “Red & green” or “manganite!”, which have different *Path Strings*. Generally, different *Path Strings* correspond to different *Perceptual Groups* as the back-end *DOM* tree structure decides how the front-end page layout looks. In other words, there is a unique mapping between *Path Strings* and *Perceptual Groups*. Such observations encourage us to make a bold statement: *Path String* may be able to **differentiate** different *Semantic Groups*. That is, for different *Text Nodes*, if their *Path Strings* are different, they are in different *Semantic Groups* (have different categories of meanings). We can even generate a one-to-one mapping between the *Path Strings* and the *Semantic Groups*.

If different *Perceptual Groups* correspond to different *Semantic Groups*, such a statement is obvious. However, this is not true for general web pages. For instance, a *Perceptual Group* in a web news portal may contain too many different items, which are in very different *Semantic Groups*. But if we limit our attention to only *DRPs*, then such a statement may be partially true for the structured or semi-structured web pages, which raises the possibility of using *Path Strings* to identify different *Semantic Groups*. When we tested this on several structured web sites, we found that for some web sites, different *Path Strings* mostly correspond to different semantic groups. For example, for social media-sharing sites such as Flickr, in any *DRPs* (as shown in Figure 9), different semantic groups such as “Tags”,

“Titles”, “Comments”, “Uploaders” all correspond to different *Path Strings*, and different “Tags” in the same *DRP* share the same *Path String*. There is a small exception of “Sets” and “Pools”. Although they have the same *Path String*, they are in different *Semantic Groups*.

In another group of web sites that we tested, we found that the *Path String* itself is not enough to differentiate semantic groups. For example in Figure 10, the book’s “Publisher” and “ISBN” are in different *Semantic Groups*, but they happen to share the same *Path String*. In this case, there is no unique mapping between a *Path String* and *Semantic Groups*. However, we will use several heuristics to identify the *feature pairs*, and this will allow us to extract different features accurately using *Path Strings* plus some additional information from the *DOM* tree. We will explain the *feature pairs* in section 5.

4.3 Path String and Path String Node Value Pair

Before we discuss *DRP* identification and data extraction, we need to define two more terms.

Definition 4: A *Path String Node Value (PSNV)* pair $P(ps, nv)$ is a pair of two text strings, the *Path String* ps , and the *Node Value* nv whose *Path String* is ps . For example, in Figure 11, “ $\langle h1 \times td \times tr \times \dots \times body \times html \rangle$ ” and “Red & green” are a *PSNV* pair.

Definition 5: We say that a *Path String*, ps , or a *PSNV* pair, $P(ps, nv)$, is **unique**, if it occurs only once in a web page’s *DOM* tree.

With these terms and definitions, we are ready to discuss our *DRP (Detail Record Page)* identification process and data extraction procedure.

5 DRP Identification

As stated in section 3, we are trying to find the pages that are *DRPs*. Our identification procedure is based on record title and record features.

5.1 Record Title Uniqueness and Cohesiveness

Our first heuristic for *DPR* identification is based on the record title and the following two observations.

Observation 1: A record title’s corresponding *Path String*, T_{ps} , is *unique*, that is, there is one unique non-empty text node value, T_{nv} , whose *Path String* is T_{ps} , and T_{nv} is the record title.

This observation is based on the fact that we are only interested in *DRPs* that contain one data record. That data record’s title must be very prominent for web browsers to identify, and its display pattern will not be repeated by other *Text Nodes*, otherwise it will not be a *DRP*. Hence the record title constitutes one unique *Perceptual Group* and *Semantic Group*. In this case, this is a one-to-one mapping between the title *Path String* and the title *Semantic Group*.

Observation 2: A web page title and record title share high similarity in *DRPs*.

This observation can be verified empirically from many popular web sites. For example, in Figure 11, the page title is “Red & green on Flickr”, while the record title is “Red & green”. However, in Figure 10, if we consider the record title as the book’s name, then the record title is not similar to page title, but if we consider the record title as “Book Chapter”, then it has high similarity with the web page’s title.

The web page title is very easy to extract as the title’s immediate parent node in the *DOM* tree is always “< title >”. However, a record’s title is hard to extract because we don’t even know whether the page, P , is a *DRP* or not.

Our first algorithm will determine the maximum Jaccard similarity between a page P ’s title and all the *Text Nodes* with *Unique Path Strings*.

algorithm 1 Extract Max Similarity Value Feature.

Input: DOM tree T of Page W

Output: Max Similarity between Page Title and all
other Text Node Values with Unique PS;

Steps

```
1 Get Page Title Title( $W$ ) from  $T$ 
2 Travs  $T$ , get all PSNV pairs,  $P_i(ps, nv), i=1, \dots, n$ 
3 From  $P_i(ps, nv), i=1, \dots, n$ , get all PSNV pairs
4 with Unique PS,  $PU_j(ps, nv), j=1, \dots, m$ 
5 Max = 0
6 for  $k=1$  to  $m$  do
7   Sim( $k$ )=Jaccard(NV( $k$ ) and Title( $W$ ))
8   if Sim( $k$ )>Max
9     then Max=Sim( $k$ )
10 return Max;
```

In Algorithm 1, getting the *PSNV* pairs in line 2 is linear to the number of nodes, N , in the tree, or $O(N)$. If we use a hash table [12], then getting the *PSNV* pairs with *Unique* PS in line 3-4 is linear to the number of *PSNV* pairs, n . Line 5-8 calculate the similarity between the page title and all *text nodes* with *unique* PS and record the maximum value. It is linear in the number of such *text nodes*, m , if we consider calculating the Jaccard similarity of short strings as constant-time complexity, then in total, the complexity is $O(N)$.

Our intuition is that if the maximum similarity value returned by Algorithm 1 is very high, then there is a good chance that the node is the record title node and the page is a *DRP*. On the other hand if the maximum similarity value is very low, say even zero, there is a high chance that the page is not a *DRP*. This intuition is based on the above mentioned observation.

However, if the record title is very short and the page title is rather long, then the maximum similarity will be low, but we can not rule out that the page is a

DRP. To address the insufficiency of this heuristic and improve accuracy, we use another heuristic, *Feature Pair*, to deal with the description part of a *DRP*.

5.2 Record Feature Pairs

First we will give the definition of a *Feature Pair*.

Definition 6: A *Feature Pair* is a pair of strings, $P(l, f)$, with feature description label l , and feature description f .

For record description or features of a *DRP*, we have the following observations.

Observation 3: A *DRP* have listed descriptions. These descriptions' corresponding *Path Strings* or *Perceptual Groups* have multiple non-empty corresponding text nodes. Each of these text nodes corresponds to one specification of the record. For example, a media (image or video) may have multiple tags. A book or an article in a Digital Library may have multiple descriptors (ISBN, Publisher, Authors, etc), or references. A product in an e-commerce site may have a list of features. Then how do we identify these descriptions? To this end, we have another observation.

Observation 4: On a *DRP*, a *Semantic Group* of listed descriptions will have only one description label. The description label's *PSNV* (*Path String Node Value*) pair is *unique* in the *DRP*. For instance, image or video tags may have "Tags" as their description label and that label's *PSNV* pair is *unique* in the *DRP*. Product description may have "features", "item-specifics", "product details", etc, as their description label. Books or articles in Digital Libraries may have "references", "related items", etc., as description label and these labels' *PSNV* pairs are all *unique*, because otherwise, we would have multiple such labels, meaning that they are not *DRPs*, and they may be *record list pages* instead.

More specifically, we have two modes of occurrences of *Feature Pairs* as shown in Figure 12. *Mode 1* is characterized by a feature label followed by a *group* of different feature descriptions, while *Mode 2* is characterized by a feature label followed by one feature description.

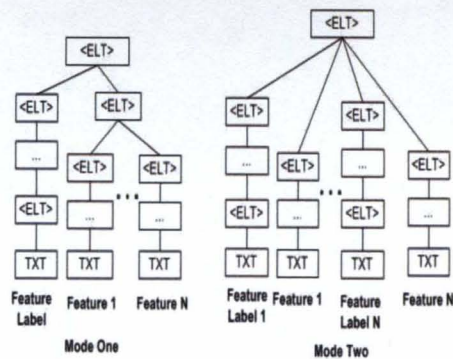
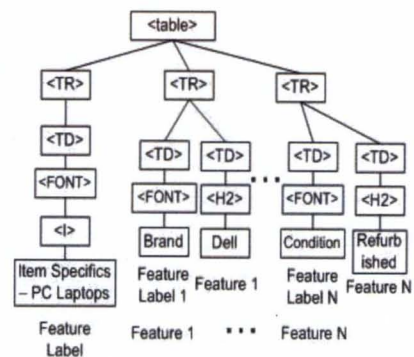


Figure 12: Two Modes of Feature Pairs.



(a) Ebay Product page



(b) Feature Pairs

Figure 13: Example Feature Pairs

Figure 13 gives the *Feature Pair* occurrence modes of an eBay product page. In Figure 13, both modes occur.

With the two types of *Feature Pair* occurrence modes available, we are now ready to present Algorithm 2, whose subroutine *getFeaturepairsWithPS* is shown in Algorithm 3.

algorithm 2 Extract *Feature Pairs* Feature.

Input: DOM tree T of Page W

Output: Feature Pairs $F_{Pi}(l,f)$, $i = 1, \dots, n$

Steps

```
1 Travs  $T$ , get all PSNV pairs,  $P_i(ps,nv), i=1, \dots, n$ 
2 From all PSNV pairs, get all PSNV pairs whose PS
3 corresponds to multi NVs,  $PM_j(ps, nv)$ ,  $j=1, \dots, m$ 
4 for  $k=1$  to  $m$  do
5    $F_{Pk}(l,f) = \text{getFeaturepairsWithPS}(T, PM_j.PS)$ 
6 return  $F_{Pi}(l,f)$ ,  $i=1, \dots, n$ 
```

In Algorithm 2, lines 2-3 get all the *Path Strings* which correspond to multiple node values. This is due to the fact that record features seldom occur alone. They often occur multiple times in a *DRP*. Line 4-5 call a subroutine *getFeaturepairsWithPS* to get *Feature Pairs* given these *Path Strings* that have multiple text nodes, or *PSM*.

First we define two general rules for feature labels, then we will explain the subroutine Algorithm 3.

Rule 1: Feature label can not be page title.

Rule 2: There cannot exist other non-empty node blocks between features and feature labels. We only extract the first pair in case that several features share the same label (mode 1 in Figure 12, as other pairs can be easily deducted).

For a given *PSM*, not necessarily all the text nodes that correspond to the *PSM* are description features. Only text nodes that satisfy the two modes shown in Figure 12 and observation 3 and 4 are considered as description features.

algorithm 3 Subroutine *getFeaturepairsWithPS*.

Input: DOM tree T of Page W , PathString PS

Output: Feature Pairs $FP_j(l, f)$, $j = 1, \dots, m$

Steps

```
1  for each Txt Node  $N_i$  in  $T$  with  $PS(N_i)=PS$  do
2    Node  $Pa=N_i$ ,  $Ch=N_i$ 
3    while  $Pa \neq NULL$  and  $Pa \neq Root(T)$  do
4      Node  $U_l=Pa.getPrevSiblingNode$ 
5      while  $U_l \neq NULL$  and  $U_l.numOfChildren()=1$ 
6        do  $U_l=U_l.getChildNode()$  //traverse down
7      if  $U_l=NULL$ 
8        then  $Pa=Pa.getParentNode()$ 
9          continue //go to upper level
10     if  $U_l.numOfChildren()>1$ 
11       then break //no feature label
12     if  $U_l=TextNode$  and
13        $U_l.getPathString() \neq Ch.getPathString()$ 
13       then if  $PSNV(U_l, PS(U_l))$  is Unique
14         then add  $FP(U_l, Ch)$  to the Results
15         else break
16      $Pa=Pa.getParentNode()$ 
17 return Results
```

Subroutine *getFeaturepairsWithPS* in Algorithm 3 is to traverse upward and downward to find the *Feature Pairs*. In Algorithm 3, the while loop in line 3-16 do the actual work. Line 4 is to try to find current text node's previous sibling or current node's parent's previous sibling (uncle). Once we find an uncle that satisfies observation 4 and rule 2 ($numOfChildren = 1$ in line 5), we traverse down the tree to get the feature label (line 6). Line 10-11, and line 15 are all to satisfy observation 4 and rule 2. If the current node's uncle node does not exist (line 7), or we could

not find the *unique PSNV* pairs in current level (line 12-13), then we traverse up one level (line 8, 9, and 16) until we are exhausted up to the tree root. The complexity of subroutine Algorithm 3 depends on the back-track (upward) and forward track (downward) to find the feature labels, with the average complexity $\text{Log}(N)$ (the worst case is to traverse two times the tree height). If we set some extra rules such as traverse past at most two “< *table* >” labels, we can save more time, as generally a description label would not be two “< *table* >”s away from the description. Thus the total complexity for Algorithm 2 is $O(N + M \times \text{Log}(N))$.

In this part, we get another feature *Feature Pairs*. Together with the title heuristic, we are ready to use a classifier to classify a page.

5.3 DRP page classifier

Up to now, given a web page, P , we have obtained the following features: (1) The maximum similarity between the page’s title and all of its *Unique Path String* text nodes, max_sim . (2) The number of *Feature Pairs*, fp_num . Because *DRPs* from different web sites may have different number of *Feature Pairs*, we normalize this feature by the total number of non-unique *PSNV* pairs, thus the feature becomes, $\text{fp_norm} = \text{fp_num} / (N_{PSNV} - N_{\text{uniq}})$. At the same time, we will use three other features to form the input for a page classifier as follows. (3) The ratio between the total number of *Path Strings* and the total number of *PSNV* pairs, $\text{ps_ratio} = N_{ps} / N_{PSNV}$. (4) The ratio between the number of *Unique Path Strings* and the number of total *Path Strings*, $\text{uniq_ratio} = N_{\text{uniq}} / N_{\text{total}}$ (5) Finally, the maximum number of *Text Nodes* a *Path String* corresponds to, max_num . The reason for adding feature (3), (4), and (5) is to help between detail pages and list pages.

With these 5 input features, we train a C4.5 decision tree classifier to classify *DRPs*.

6 DRP Data Extraction Based on Path String

In section 5, we used several heuristics to *identify DRPs*, now we move our attention to how to extract useful data from *DRPs*.

6.1 Structured Organization of DRPs

We have discussed the structured organization of DRPs for different web sites. For example, if we visit Flickr.com, we will find that all the detail image pages are organized in a similar fashion: One big image, with image title, image uploader, and image tags, etc. Moreover, all these features are in the same fixed position in a specific detail page. This phenomenon is not accidental. It is due to the way by which these web pages are designed and maintained. All these pages are generated by one single template with different data being inserted to fixed positions, it would thus be strange if they did not show similar layout patterns.

6.2 Path String based Data Extraction

In chapter II, we reviewed several methods for extracting data from structured web sites. All these methods use very complicated steps and heuristics, struggling to deduce the whole template of the web pages accurately. One thing that needs to be pointed out is that the ultimate purpose of accurate template deduction is *extracting the target data*. As long as we can extract our target data accurately, we don't even need to deduce the whole template into such details as to single character matching. In the following, we will present our *Path String* based method for data extraction from structured web sites.

Depending on how much accuracy we want and how our target website is organized, we have two scenarios.

Scenario 1: Single Specific Website DRP Extraction

This scenario works for a single specific website, say, if we only want to extract image or video textual information from flickr or youtube. In this scenario,

we use human manual assignment, which is very simple, to *name* feature groups that we desire. This scenario works in the following way: first, we manually select a training page (*DRP*) from the target website, such as the page shown in Figure 9. Then, we use the system to output all the *Path Strings* that correspond to all the *Text Nodes* in the *DRP*. After that, we manually identify the nodes for the attributes we want to extract and use their *Path Strings* to build the template schema. Finally, we use this schema to parse all the other *DRP* pages. For example, in the Flickr image page case, using our extraction rules and after validation, we find that the *Path String* for image title is: “<h1<td<tr<table<div<body<html>”. We will use this *Path String* setting to parse all the remaining image detail pages. In the parsing process, we will categorize any *Text Node* whose *Path String* is “<h1<td<tr<table<div<body<html>” as title node. We apply a similar procedure to other attributes such as image uploader, tags, comments, etc.

Special Cases for Scenario 1

1) Optional Data Using this method, the question of optional data is trivial. If an item is optional and does not occur in one specific page, the method for retrieving that type of *Text Nodes* with the *path string* of the same type will return null. For example, some pages may have comments while others do not, when a page doesn’t have any comments, we would get a null when using *comments path string* to retrieve the comments in that page. All we need to do is to enlist all the attributes of our interest and extract *path strings* for them regardless of whether they are optional.

2) Disjunction For the case of disjunctions, we need a step further to differentiate specific types. For instance, for Flickr image page, the group (pool) and set attributes have the same *Path Strings*. They can be thought of as a generalization of disjunctions as they can both occur, both not occur or occur exclusively in the same page. We can use the textual information of the items to differentiate these two types (A pool contains the keyword “pool” while set contains the keyword “set”).

3) Template Text Some texts occur in almost every page and form *frequent pattern texts*. Generally, we don't need to worry about these texts as there is little chance that they share the same *path string* with the target texts that we need to extract. But in case there is an overlap of *path strings* between that of the *frequent pattern texts* and that of the target texts, we can easily prune such *frequent pattern texts* solely because their occurrence frequency in the total collection is higher than a pre-set threshold. We will discuss this schema generation process later in scenario 2.

We can see in this scenario that the *Path String* method can handle these special cases without engaging much extra effort. As can be seen from the above explanation, compared to methods mentioned in chapter II, our method is very straightforward and effective. In the following we will discuss a more involved scenario, applying *Path String* method to the whole web.

Scenario 2: Scalable Automatic Extraction

If we want to use our method for many more websites, then the human intervention procedure in scenario 1 would no longer scale. For this scenario, instead of extracting only useful information actively as in scenario 1, we need to remove the template data to achieve useful data extraction. That is, we need to build the schema as most data extraction methods would do. For this scenario, the schema deduction process is given in Algorithm 4.

algorithm 4 Deduce Schema from DRPs.

Input: N DRPs for schema extraction

Output: schema PSNV-pairs, $P_i(nv, PS(nv))$, $i=1, \dots, n$

Steps

- 1 Schema = All PSNV-pairs of Page 1
 - 2 for $i=2$ to N
 - 3 do Temp=All PSNV-pairs of Page i
 - 4 Schema=intersection(Schema, Temp)
 - 5 Return Schema.
-

In Algorithm 4, we adopt a simple way of identifying schema data and real data. That is, if the data value and its path string pair (*PSNV* pair) occur in every page, we identify it as schema pair, otherwise it is a real data pair. If there is only one page as input, then the whole page's path string node value pairs are schema data. The for loop of lines 2-4 does a simple intersection operation of the pages. Line 5 returns the schema. Note that this is a simple intuitive way of generating schema, that can be expanded later, by setting a threshold value. So if a certain *Path String Node Value* pair occurs in a certain percent of the total pages, then it will be identified as schema data. In our experiments, we found that with this simple approach, we can generate schema which are comparable to other related methods.

6.3 Case study: applying DTPS methods on Flickr

Flickr has many features to organize the images. These include: image *titles*, *uploaders*, *tags*, *sets*, *groups* (*pools*), *comments*, etc. As shown in Figure 9, the image with title *Red & green* belongs to different *groups* (*pools*: *Kris Kros ...* etc.) and *sets* (*germany* etc). HTML 1 gives the shortened HTML code after removing irrelevant content (some *img*, *script*, *style*, and *template text* etc.) for the flickr page shown in Figure 9.

code 1 Sample Flickr Page Code.

```
<html><head><title>Red &amp; green on Flickr</title>
</head><body><div><table><tr>
  <td><h1>Red &amp; green</h1></td>
  <td><div>Uploaded on <a>January 17, 2007</a><br />
    by<a><b>manganite</b></a>
  </div><div><p>This photo also belongs to:</p>
    <div><table><tr><td><h3><a>
      germany (Set)</a></h3></td></tr></table></div>
    <div><table><tr><td><h3><a>
      flowers (Set)</a></h3></td></tr></table></div>
    <div><table><tr><td><h3><a>
      Utata (Pool)</a></h3></td></tr></table></div>
    <div><table><tr><td><h3><a>
      The World ... (Pool)</a></h3></td></tr></table></div>
    <div><table><tr><td><h3><a>
      Kris Kros ... (Pool)</a></h3></td></tr></table></div>
  </div><div><h4>Tags</h4><div>
    <div><a>europe</a></div>
    <div><a>germany</a></div>
    <div><a>nrw</a></div>
    <div><a>herten</a></div>
  </div></div></td>
</tr></table></div></body></html>
```

Figure 14 gives the DOM tree structure for the page in HTML 1 code.

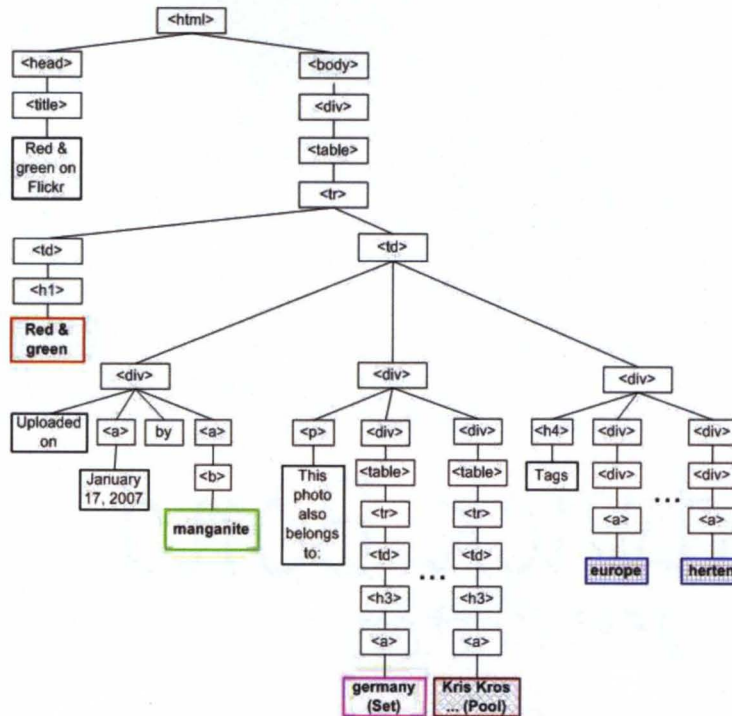


Figure 14: Flickr Image Page DOM tree for Figure 9.

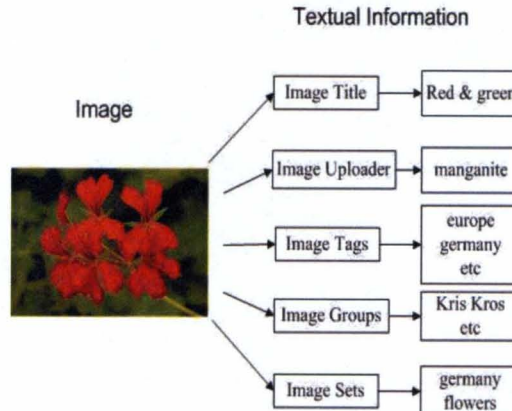


Figure 15: Extracting Textual Information from a Flickr Image Page.

We can see that some *semantic groups* form the DOM tree structure. For example, “europe” and “herten” are in the same *semantic group* as they all belong to tags. Also we can see from the DOM tree that different *semantic groups* have different *path strings*. For instance, the *path string* for title “Red & green” is different from the *path string* for image uploader “manganite”, which are all

different from the *path string* of image sets “germany”, etc. Finally, Figure 15 shows the extraction results using our *DOM tree path string* (DTPS) method.

7 Summary

In this chapter, we have presented a simple DOM tree *Path String* based methods for *DRP* identifications and data extraction. We have adopted several heuristics to identify the *DRP* patterns from web pages based on record title and page title similarity and record *Feature Pair* occurrences. We also presented two scenarios for applying our *Path String* based method for real web record extraction. In the following chapters, we are going to incorporate the *DRP* classification scheme and *Path String* based data extraction method for scalable web image data record crawling.

CHAPTER IV

Focused Image Crawling

1 Introduction

Social media-sharing web sites, such as Flickr, YouTube, etc., are becoming more and more popular. These web sites not only allow users to upload, maintain, and annotate images/videos, but also allow them to socialize with other people through contacts, groups, and subscribers, etc. Two types of information are generated in this process. The first type of information is the rich text, tags and multimedia data uploaded and shared in such web sites. The second type of information is the users' profile information, which tells what kind of topics interest them. Research on how to use the first type of information has been brought into attention recently. However, little attention has been paid to effectively exploit the second type of information, which are the users' profiles, for improving image search.

On the other side, the concept of *vertical search engine* and focused crawling has come into focus gradually against popular-based general search engines. Comparing with general search engines, as the topical search engines only focus on specific areas, they are more likely to become experts in such areas. Although they lack the broadness that general search engines have, their depth can win them a stand in the competition.

In this chapter, we are going to explore the applicability of developing a focused crawler on social multimedia web sites for better search. More specifically, we are going to exploit the users' profile information from social media-sharing web sites for developing a focused crawler to better serve people's needs for accurate multimedia search. To begin the focused crawling process, we need to first accurately identify the right type of pages. To this end, we propose to use a *DOM*

path string based method for page classification, which was discussed in the previous chapter. Correct identification of the right type of page can not only improve crawling efficiency by skipping undesirable types of pages, but also helps improve the accuracy of extracting data from these pages. In addition, we use a co-tagging method for topic discovery as we think it suits multimedia crawling more than the traditional taxonomy methods do (For instance, it is more flexible to new topics and new media additions).

This chapter is organized as follows. In section 2, we define three types of pages and discuss our focused crawling motivation. In section 4, we introduce our profile-based focused crawling policies. In section 5, we discuss the co-tagging topic discovery for the focused crawler. In section 6, we present the focused crawling process.

2 Motivation for Profile-based Focused Crawling

2.1 Popularity of Member Profile

In chapter II, we reviewed several focused crawling systems. These focused-crawling systems analyze the probability of getting pages that are in their crawling topics based on these pages' parent pages or sibling pages. In recent years, another kind of information, members' profiles, started looming large in social networking and resource-sharing sites, and almost eludes all current focused crawling efforts. We will explore the applicability of using such information in our focused-crawling system. More specifically, to illustrate our profile-based focused-crawling, we will use Flickr as an example. But our method can be easily expanded to other social networking sites, photo-sharing sites, or video-sharing sites, hence we refer to them as "social multimedia websites".

2.2 Typical Structure of Social Media-sharing Websites

Social media-sharing Websites, such as Flickr and YouTube, are becoming more and more popular. Their typical organization structure are through these

different types of web pages defined below.

1. A **List Page** is a page with many image/video thumbnails and their corresponding uploaders (optionally some short descriptions) displayed below each image/video. A *list page* can be considered as a *crawling hub page*, from where we start our crawling. An example *list page* is shown in Figure 16.



Figure 16: An Example List Page on Flickr



Figure 17: An Example Detail Page on Flickr

2. A **Detail Page** is a page with only one major image/video and a list of detailed description text such as title, uploader, tags, etc around it. A *detail page* can be considered as a *crawling target page*, which is our final crawling destination. An example *detail page* is shown in Figure 17.
3. A **Profile Page** is a page that describes a media uploader's information. Typical information contained in such a page includes the uploader's

image/video sets, tags, groups, and contacts, etc. Further, such information can be divided into two categories: **inner properties**, which describe the uploader's own contributions, such as the uploader's photo tags, sets, collections, videos, etc., and **inter properties**, which describe the uploader's networking with other uploaders, such as the uploader's friends, contacts, groups, subscribers, etc. We will use information extracted from *profile pages* to guide our focused crawling process.

A *list page* has many outlinks that point to *detail pages* and *profile pages*. The structure is shown in Figure 18, in which two image thumbnails in a *list page* link to two *detail pages* and corresponding *profile pages*.

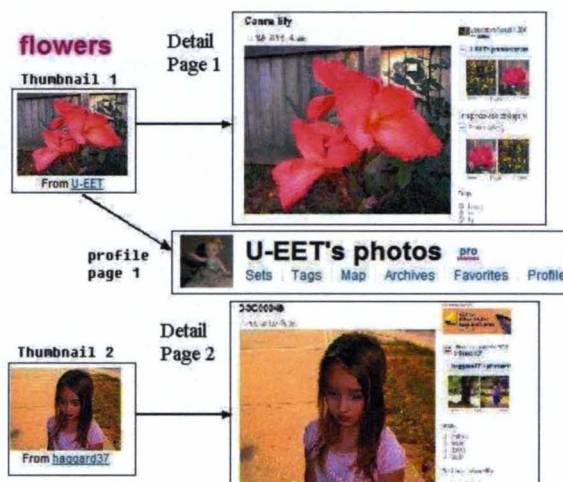


Figure 18: Typical Structure of List Page, Detail Pages, and Profile Pages.

2.3 Profile-based Focused-crawling

Our motivation while crawling is to differentiate the importance of each outlink or *detail page link* before we actually retrieve that *detail page* given a *list page* and a crawling topic. For the case of Figure 18, suppose we are going to crawl the topic of *flowers*, our intuition would rank the first *detail page link*, which links to a real flower, higher than the second *detail page link*, which links to a walking girl and happened to be also tagged as “flower”. The only information available for us to use is the photo thumbnails and the photo uploaders such as “U-EET” and

“haggard37”, etc. Processing photo thumbnails to recognize which one is more conceptually related to the concept of real *flowers* poses a challenging task and we leave it as future work. We will explore the photo uploader information to differentiate different concepts. Luckily, most social media-sharing Websites keep track of each member’s profile. As shown in Figure 18, a member’s profile contains the member’s collections, sets, tags, archives, etc. If we process all this information first, we can have a preliminary estimate of which type of photos the member would mainly upload and maintain. We can then selectively follow the *detail page links* based on the corresponding uploader profiles extracted.

3 Path String based Page Classification

Before we actually do the crawling, we need to identify the type of a page. In this section, we will discuss our page classification strategy based on DOM *path string* method. Using such method, we are able to identify whether a page is a *list page*, *detail page*, *profile page*, or none of the above.

3.1 Classifying Pages based on Real Data Path Strings

Note same types of pages have the same *perceptual groups* and further same *path strings*. We then can use whether a page contains a certain set of *path strings* to decide whether this page belongs to a certain type of pages. For example, as we already know that all *list pages* contain the *path string* that corresponds to *uploader names*, and almost all *detail pages* contain the *path string* that corresponds to *tags*, we can then use these two different types of *path strings* to identify *list pages* and *detail pages*. Algorithm 5 gives the procedure of extracting characteristic *path strings* for a type of pages.

algorithm 5 Extract Page Path Strings.

Input: N Pages of the same type

for page type path strings extraction

Output: A Set of Path Strings, PS_i , $i=1, \dots, n$

Steps

1 Set = All Path Strings of Page 1 - Schema PSs.

2 for $i=2$ to N

3 do Temp=All PSs of Page i - Schema PSs

4 Set=intersection(Set, Temp)

5 Return Set.

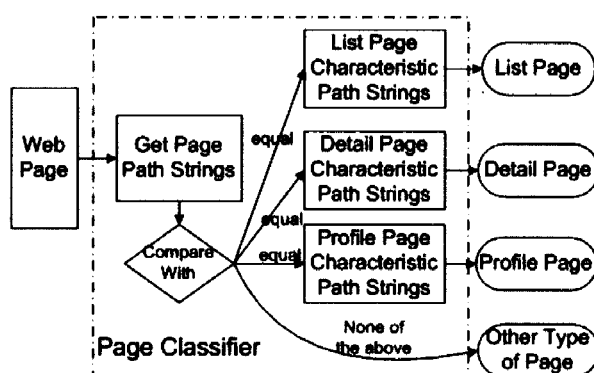


Figure 19: Page Classifier

By applying Algorithm5 on each type of pages (*list page*, *detail page*, and *profile page*), we are able to extract a group of characteristic *path strings* for each type of pages. Then given a new page, the classifier would only need to simply check whether that page contains all the *path strings* for a group to decide whether that page belongs to that type of page. The process is shown in Figure 19.

Note that most of the time, we don't even need to compare the whole group of *page path strings* with *characteristic path strings*, a few typical *path strings* would suffice to differentiate different types of pages. For example, as we tested on flickr, we find that only one *path string* for each type of page is enough to do the

classification.

4 Profile based Focused Crawler

Using the *path string* method, we are able to identify the correct page types. In this section, we are going to discuss our profile-based crawling system. The basic idea is based on an uploader's profile, we have a rough understanding of the topic interests of the uploader, then when we encounter an image/video link of that uploader, we will have a prior knowledge of whether that image belong to our crawling topic and we process that link accordingly. By doing this, we are able to avoid extracting the actual *detail page* to know whether that page belongs to our crawling topic. To this end, we further divide the user profile into an *inner profile* and an *inter profile*.

4.1 Ranking from the Inner Profile

The inner Profile is an uploader's individual property. It comes from the uploader's general description of the media that they uploaded. From that, we can roughly identify the type of this uploader. For instance, a nature fan would generally upload more images and thus generate more annotations about nature; an animal adorer would have more terms about animal, dog, pets, etc., in their profile dictionary. For Flickr photo-sharing, an uploader's *inner profile terms* come from the names of their "collections", "sets", "tags". For YouTube page, an uploader's *inner profile* comes from their "videos", "favorites", "playlists", etc.

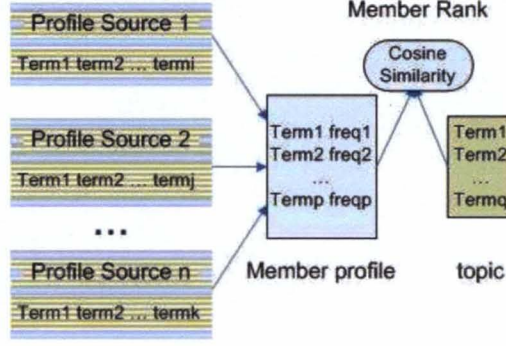


Figure 20: Inner Profile Ranking

The process for calculating *inner profile rank* can be illustrated with Figure 20. After we collect all the profile pages for an uploader, we extract terms from these pages, and get one final profile *term vector*. We then calculate the *cosine similarity* between the profile *term vector* and the topic *term vector* to get the member's *inner profile rank*. We use Equation 9 to calculate a user's inner rank.

$$Rank_{inner}(u|\tau) = Cos(\vec{x}_u, \vec{x}_\tau) \quad (9)$$

where \vec{x}_u is the term vector of the user, and \vec{x}_τ is the topic term vector.

4.2 Ranking from the Inter Profile

The inner profile gives only an uploader's individual properties. However, since an uploader is in a social media-sharing websites and he/she socialize with other uploaders in the same site, we need to take such social networking activities into consideration. The motivation behind this is if a user is a big fan of one topic, then he will tend to have friends, contacts, groups, or subscriptions, etc., which are related to that topic. Through social networking, different uploaders form a graph. However, such a graph is very sparse. An uploader may have only a limited number of social contacts. Hence, it is hard to conduct a systematic analysis on such a sparse graph. In this paper, we will use a simple method, in which we just accumulate an uploader's social contacts' *inner ranks* and sum them up to get the uploader's *inter rank*.

Suppose a user u have N contacts, then the *inter rank* of the user is calculated with Equation 10.

$$Rank_{inter}(u|\tau) = 1/N \sum_{i=1}^N Rank_{inner}(c_i|\tau) \quad (10)$$

where τ is the given crawling topic, and $Rank_{inner}(c_i|\tau)$ is the user's i^{th} contact's *inner rank*.

4.3 Combining The Inner-Rank and Inter-Rank

For focused crawling, our final purpose is to find the probability of following link L_n given the crawling topic τ so that we can decide whether we should follow the link. Using Bayes Rule, we have:

$$Pr(L_n|\tau) = \frac{Pr(\tau|L_n) * Pr(L_n)}{Pr(\tau)} \quad (11)$$

Suppose there are N total candidate links, then

$$Pr(\tau) = \sum_{0 < i \leq N} (Pr(\tau|L_i) * Pr(L_i)) \quad (12)$$

Our task is then transformed into calculating the prior $Pr(\tau|L_n)$, that is, given a link, the probability of that link belongs to crawling topic τ . We propose to calculate the prior based on *inner ranks* and *inter ranks*. Each factor gives us a reward of following the link. We combine them together with the following equation.

$$Pr(\tau|L_n) = \alpha \times Rank_{inner}(u_m) + \beta \times Rank_{inter}(u_m) \quad (13)$$

where L_n is the n^{th} image thumbnail link and u_m is the m^{th} user that corresponds to the n^{th} image thumbnail link. $Rank_{inner}(u_m)$ and $Rank_{inter}(u_m)$ are calculated using Equation 9 and Equation 10 respectively. We can normalize them to make $Pr(\tau|L_n)$ a valid probability, but for practical applications, they are just used for ranking links to decide which ones to follow, and thus they don't have to be transformed into probability ranges.

5 Co-tagging Topic Discovery

To start the focused crawling process, we need to feed the crawler with a crawling topic. The crawling topic cannot be set only as one tag as that would be too narrow. For example, if we choose the crawling topic as “animals”, all tags that are closely related to “animals”, which may include “cat”, “dog”, “pet”, etc., need also to be included in the crawling topic tags. Hence, to set a crawling topic properly, we need to expand the topic tagging words. Our method to conduct this task is by exploiting image/video co-tagging.

We use a voting-like processing method. If one tag, say **T1**, and the topic tag **T** co-occurred in one photo, we count this as one vote of **T1** also belonging to our crawling topic. When we accumulate all the votes through a lot of photos, we would get a cumulative vote for **T1** also belonging to our crawling topic. When such a vote is above a threshold, we will include tag **T1** in our crawling topic tags. For real application, we use a correlation threshold.

$$\varphi = \frac{P(T \cap T1)}{P(T) \times P(T1)} \quad (14)$$

where $P(T \cap T1)$ is the number of pictures co-tagged by both tag **T** and tag **T1**, and $P(T)$ and $P(T1)$ are the number of pictures tagged by tag **T** and tag **T1** respectively. Suppose tag **T** belongs to the crawling topic, φ gives the score of whether **T1** also belonging to the crawling topic. When φ is bigger than a pre-set threshold, we will count **T1** as belonging to the crawling topic.

In order to make the crawling topic tags more robust, we use the following strategies.

1. Take only one image if multiple images are tagged with the same set of tags. This is usually because an uploader may use the same set of tags to tag a group of images they uploaded to save some time.

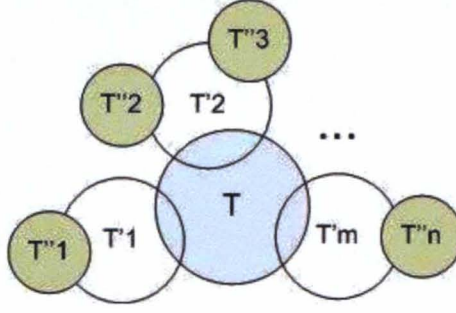


Figure 21: Two Layer Co-tagging Topic Discovery.

2. From the top co-tagging tags, start a new round of co-tagging tag discovery process. The process can be described as in Figure 21, then use the expanded cluster of high frequent co-occurent tags as the final crawling topic.

6 Profile-based Focused Crawling System

We have a two-stage crawling process that includes a co-tagging topic discovery stage and a profile-based focused crawling stage. Both of these stages use the proposed page classifier extensively to avoid unnecessary crawling. We will give the diagrams and algorithms for these two processes below.

6.1 Co-tagging Topic Discovery Stage

The first stage of our profile-based focused crawling system is the co-tagging topic discovery stage. In this stage, we collect images that are tagged with the initial topic tag, record their co-tags, process the final co-tagging set, and extract the top-frequent co-occurent ones. Figure 22 gives the diagram of the working process of this stage, and Algorithm 6 gives the detailed steps.

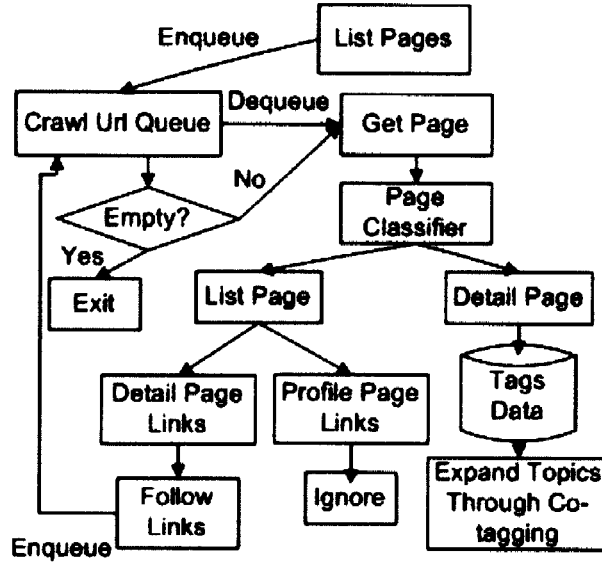


Figure 22: Stage One : Co-tagging Topic Expansion Stage

In Algorithm 6, lines 4-14 do the actual crawling work. The page classifier described in section 3 is used in line 6 to decide whether a page is a *list page* or a *detail page*. We already know that in social media-sharing web sites, *list page* have outlinks to *detail pages* and *profile pages*, and we name such links as *detail page links* and *profile page links* respectively. It is usually easy to differentiate them because in the *DOM* tree structure, *detail page links* generally have image thumbnails as their children, while *profile page links* will have *text nodes*, which are usually the uploader names, as their children. Combined with our *path string* method, we can efficiently identify such outlinks. In lines 11-12, by not following *profile page links*, we save lots of work. Since we are not following *profile page links*, the classifying result for page p in line 6 would not be a *profile page*. Lines 15-16 do the co-tagging analysis work and line 17 return the expanded topic tags.

algorithm 6 Stage one : Co-tagging topic discovery

Input: Initial Crawling Topic Tag, T

List pages, $p_1, \dots p_N$

Output: Expanded Topic Tags, $T, T_1, \dots T_k$

Steps

```
1 Set Queue  $Q$  = empty
2 for  $i=1$  to  $n$ 
3   do Enqueue  $p_i$  into  $Q$ 
4   while  $Q \neq \text{Empty}$ 
5     do page  $p$  = Dequeue  $Q$ 
6     classify  $p$ 
7     if  $p = \text{List Page}$ 
8       then  $\langle o_1, \dots o_m \rangle = \text{Outlinks from } p$ 
9         if  $o_i = \text{Detail Page Link}$ 
10           then Enqueue  $o_i$  to  $Q$ 
11         else if  $o_i = \text{Profile Page Link}$ 
12           then discard  $o_i$ 
13     else if  $p = \text{Detail Page}$ 
14       then extract tags data from  $p$ 
15 analyze the tags to get to top frequency
16 co-occurred tags  $\langle T, T_1, \dots, T_k \rangle$ 
17 return  $\langle T, T_1, \dots, T_k \rangle$ 
```

6.2 Profile-based Focused Crawling Stage

In the second stage or the actual crawling stage, we use the information acquired from the first stage to guide our focused crawler. For this stage, depending on the system scale, we can choose to store the member profile either on hard disk or in main memory.

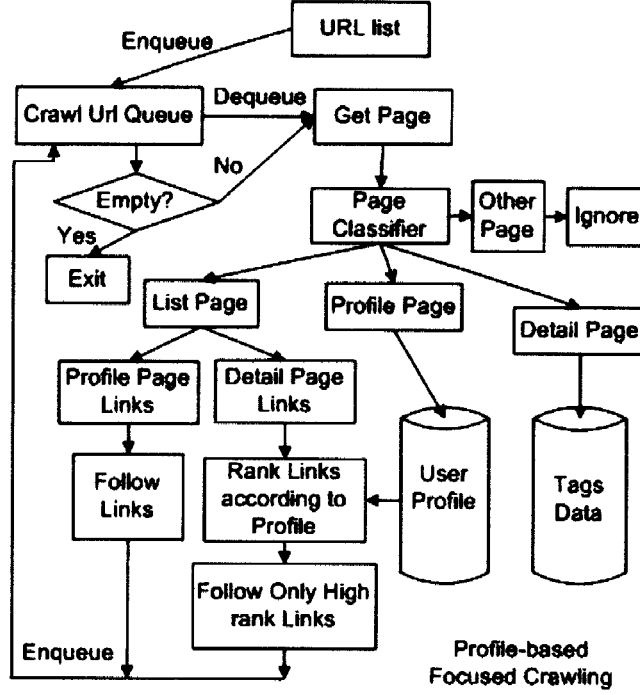


Figure 23: Stage Two : Profile-based Focused Crawling Stage

The system diagram is shown in Figure 23, and the process detail is shown in Algorithm 7. In Algorithm 7, similar as the co-tagging stage, we classify page p in line 6. The difference is, since we are not pruning *profile page links* in lines 13-14 and we follow them to get user profile information, we will encounter *profile page* branch in the classification result for line 6, as shown in lines 17-18. Another difference is how we handle *detail page links*, which is shown in lines 10-12. In this stage, we check whether a *detail page link's user profile rank* according to the crawling topic. If the rank is higher than a pre-set threshold, $RANK_{TH}$, we will follow that *detail page link*, otherwise, we will discard it. Note that in this process, we need to check whether a user's *profile rank* is available or not, which can be done easily by setting a *rank available flag*, and we omit this implementation detail in the algorithm. In lines 17-18, we process profile pages and extract profile data. Another issue is when to calculate the user profile rank since the profiles are accumulated from multiple pages. We can set a fixed time interval to conduct the calculation or use different

threads to do the job, which is another implementation detail that we will skip here.

algorithm 7 Stage Two : Profile-based Focused Crawling

Input: Crawling Topic Tags, $\langle T_1, \dots, T_k \rangle$

Crawling URLs $\langle url_1, \dots, url_n \rangle$

Output: Crawled Detail Pages

Steps

```
1 Queue Q = empty
2 for i=1 to n
3   do Enqueue url_i into Q
4 while Q != Empty
5   do page p = Dequeue Q
6     classify p
7     if p = List Page
8       then  $\langle o_1, \dots, o_m \rangle$  = Outlinks from p
9         if o_i = Detail Page Link
10           then if Rank(u_o_i) > RANK_TH
11             then Enqueue o_i to Q
12             else Discard o_i
13         else if o_i = Profile Page Link
14           then Enqueue o_i to Q
15   else if p = Detail Page
16     then Extract Tags Data from p
17   else if p = Profile Page
18     then Extract Prof Data from p
19   else if p = Other Type Page
20     then ignore p
21 Return Detail Pages Tags Data
```

7 Summary and Discussions

This chapter explained in detail how profile based image crawling works. We use inner profile and inter profile to decide which image to crawl. At the same time, some of those profile information maybe considered good candidate for ranking the image.

- a) **Image Uploader Profile.** In the query ranking stage, if we can use the query keyword to match the image uploader profiles. If we find a match between an image uploader's top tagging categories and the query keyword, that uploader's images can be ranked higher.
- b) **Image Comments.** Because comments can contain too much noise (including spam) it needs much preprocessing to be useful. However, if we can count how many comments an image has received, and use this as a ranking factor (since, in general, the more comments an image has, the more interesting it is), that would also be useful.

There are some open issues worth discussion.

1. **Image Title:** What about long titles? Are there some heuristics for extracting meaningful keywords from long sentence? What about meaningless number titles? Can we recognize these types of titles by applying some rules.
2. **Image Tags:** This is the part in which we are mainly interested. However, several problems remain: (1) more and more people tend to use the same set of tags to annotate a whole set of images. This can make the tags un-reliable; (2) Camera models and lens types are starting to enter into image tags, adding further noise.

CHAPTER V

Image Indexing

1 Introduction

As we intend to build an image search engine for the social Web, besides indexing image text (tags or annotations around the image link), we need to pay close attention to indexing image content features such as colors and textures. In this chapter, two types of image *content* will be explored: *color* and *texture*.

In section 6 of Chapter II, we have reviewed different indexing schemes used in current CBIR systems and intuitively explained why we chose to use *inverted files* for our indexing architecture. In this chapter, we will perform further experiments to justify our inverted indexing scheme. Then, we are going to present our clustering-merge algorithm that explicitly seeks to better model power-law distributions. After that, we will explain our procedure to extract these content features and to convert them into textual codewords for indexing.

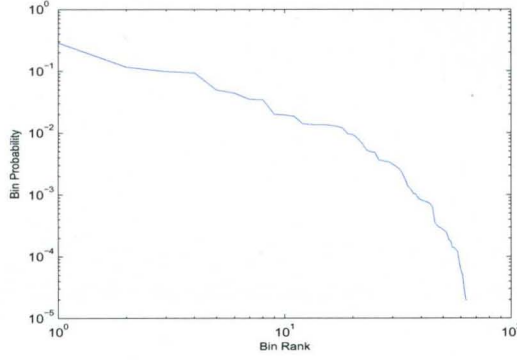
In section 5 of chapter II, we have reviewed different image processing techniques. One of the most important methods that we propose is the *tile-based gridding scheme* based on spatial image segmentation. More specifically we define three types of image tiles: *inner tiles*, *bordering tiles*, and *crossing tiles*. Through these definitions, we hope to combine effectiveness and efficiency in dealing with real world image content and thus realize a scalable image search system. We will also consider dividing an image into sub-blocks (9 blocks) for further processing. Later, in explaining our image texture extraction process, we will also discuss how we divide an image into small tiles for processing. In each step, we will pay special attention to the computational complexity and the performance bottleneck that may be present in the real-world system.

2 Image Indexing

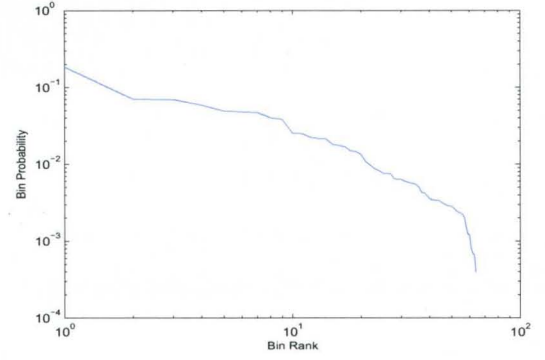
2.1 Sparseness of Image Features

One assumption and reason behind the success of the inverted file structure for indexing text and documents is the sparseness of terms in text documents. The total number of terms in a large document collection is very high with dimensionality $O(10^4)$ [59] [66], while the number of terms that occur in a single document is comparatively low $O(10^2)$ [59] [66]. In other words, human have accumulated a big enough vocabulary to cover each field of our everyday life, yet for a specific topic or article, a small portion suffices. However for the case of images, since “A picture is worth a thousand words” and since pictures tend to be self-explanatory, it can be hard to develop a complete thesaurus for images. The result is that a *sparse* dictionary comparable to documents does not exist. However, an analysis of images based on its color, texture, and shape, shows that although a term-document like high sparseness is not achieved, a slightly lower sparseness could be reached with an appropriate content to codeword mapping. For instance, a picture may contain relatively only a few dominant colors when using a color histogram containing hundreds of bins. The sparseness also depends on the visual word representation.

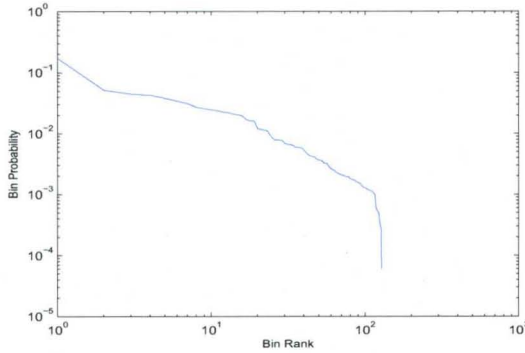
Figure 24 gives the log-log distribution of the resulting colorwords (mapped from original colors) using different methods to represent 10,000 actual images selected randomly from flickr.com. For equal color quantization, we use $4 \times 4 \times 4$ quantization on the RGB color channels to get 64 bins. For the clustering-merge based method, we use 1000 images as the training set to obtain the global color or texture codebook, then use the global codebook to represent 10,000 images randomly selected from flickr.com.



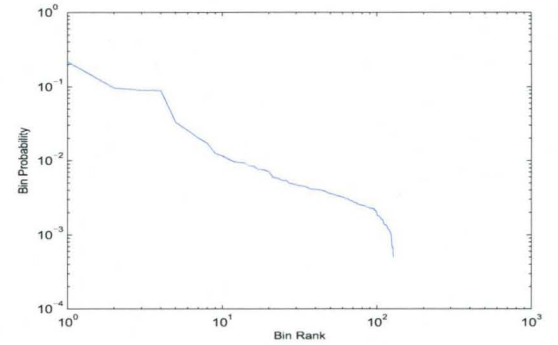
(a) Equal Quantization (Color 64 bins)



(b) Clustering-Merge Based Method (128 Colors merged into 64 centers)



(c) Clustering-Merge Based Method (256 Colors merged into 128 centers)



(d) Clustering-Merge Based Method (256 Textures merged into 128 centers)

Figure 24: Log-log plots of different quantization methods.

We can see that these representations roughly follow a power law distribution. In [35], Frederic et al showed that texture features followed a power law distribution. Their results match our experimental results, and provide a motivation for our inverted indexing scheme. In the following, we will discuss our clustering-merge method to compute the global codebook.

2.2 GLA Clustering and Cluster-Merge Algorithm

We have reviewed the Generalized Lloyd Algorithm for generating visual codebooks in section 6 of Chapter II. In this section, we will discuss why we need GLA to generate visual codebooks as an alternative to another method that we have used in [71]. First, uniform quantization is not feasible for high-dimensional

features, whereas GLA can be used in such cases. For image color codebook generation, we can choose uniform quantization since there are only three channels (R,G,and B) and the dimensionality is three, but for the Gabor texture feature, the dimensionality is high (48 or more), and we can not use uniform quantization to generate the codebook, so we have to rely on clustering, for example, to generate the codebook. The second reason why we use GLA to replace uniform quantization is that the latter does not take the actual feature distribution into account and can lose some accuracy in representing the images.



(a) Equal Quantization



(b) Clustering-Merge Based Quantization

Figure 25: Quantization Results Comparison (64 bins).

Figure 25 compares the quantization of the picture shown in Figure 30. We used a global codebook for the clustering case. We can see that clustering merge based quantization gives more smooth and better quantization results. Because clustering tends to choose better centroids.

One issue with using most k-mean-based clustering algorithms such as GLA for codebook generation is that for power law distributed data, more cluster centers are generated around high density areas [35]. This can cause the codewords to be overpopulated in the high density area while the sparse area are not properly represented. To solve this problem, we propose to merge these very close cluster centers after obtaining the K-means clustering result. We first use K-means to generate more centers than what we expect, then we merge these centers into the number of centers we want. Through merging close centers, we can discourage cluster centers from crowding into high density areas. At the same time, we fully take

advantage of the efficiency of K-means. Thus, our clustering merge algorithm will be much more efficient than purely a hierarchical clustering algorithm. The pseudocode for merging clusters is shown in algorithm 8.

algorithm 8 Pseudocode for “cluster-merge”.

Input: n clusters

Output: k clusters

MergeCluster(n, k, r)

if($k \geq n$)

 return n ; //already less than k clusters

while($k < n$)

 do

 find two closest centers c_1, c_2 among n clusters

 if($\text{dist}(c_1, c_2) > r$) //radius limit

 break;

 merge c_1, c_2 into c

 insert c into the list of cluster centers

 delete c_1 and c_2 from the list of cluster centers

$n = n - 1$

return n

In algorithm 8, we check two constraints, the number of clusters and the distance between two closest clusters. Once we obtain the desired number of clusters or if the distance between two closest clusters becomes greater than the threshold distance, we terminate the merging and return the results.

2.3 Building an Inverted Index for Image Content with “Field Boosting”

In a similar way that we build an inverted index for documents, we can build an inverted index for images by using the image visual *code-words*. We will discuss the process of how to generate these *code-words* in Section 3.

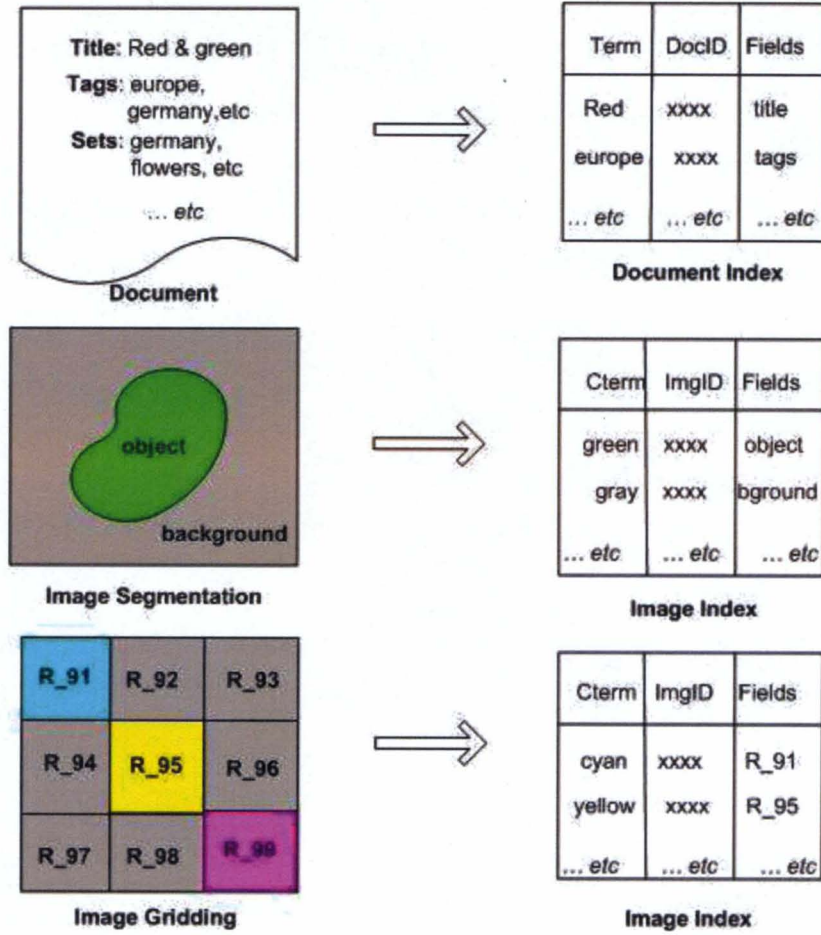


Figure 26: Indexing an Image Using Inverted File.

As shown in Figure 26, in the same way that we can divide a web page document into different fields (blocks) like title, text, url, etc and record these fields in the inverted file for further retrieval, we can divide an image into different fields. When using image segmentation, we can mark different fields like object, and background, etc. Similarly, when we use gridding to divide the image, we can mark the resulting regions as different fields such as R_91 and R_92.

Document or field *Boosting* [27] is a way of selectively boosting documents or fields so that they receive a high relevance score in matching. In our case, we can assign different *boosting factors* (α) to different “fields” or blocks of an image as follows.

$$\alpha_{image\ field}(block) = \frac{1}{d(block, center) + 1} \quad (15)$$

where d is the average distance of the image “field” (grid, block, or segment) to the image center. For instance, if we impose a grid on the image with 9 blocks as shown in Figure 26, the average Manhattan distance for block R_95, R_92, R_91 can be set as 0, 1, 2 respectively, which correspond to a *boosting factor* of 1, $1/2$, and $1/3$ accordingly.

Building an inverted index for images in such a way will not only help us emphasize the important image regions, it can also help us in retrieving user designated query regions. For instance, in Figure 26, if object regions are to be considered more important than background regions, then and R_95, the center block, may deserve more attention for the final weighting than R_91, the border region. In the query interface, by allowing users to choose which region to have certain characteristics to search, we can better serve users’ needs in filtering irrelevant images. We can also increase the sparseness of the content code words, as mentioned above.

3 Image Codebook Design and Feature Extraction

In this section, we will discuss how we convert the image content features such as color and texture, into text for indexing. In section 6 of chapter II, we have reviewed the GLA or K-means algorithm. In section 2.2, we discussed our cluster-merge method for better codebook representations. In the following, we are going to use the cluster-merge algorithm in image codebook design, by explaining the detailed steps for dealing with each image feature.

3.1 Image Color Codebook

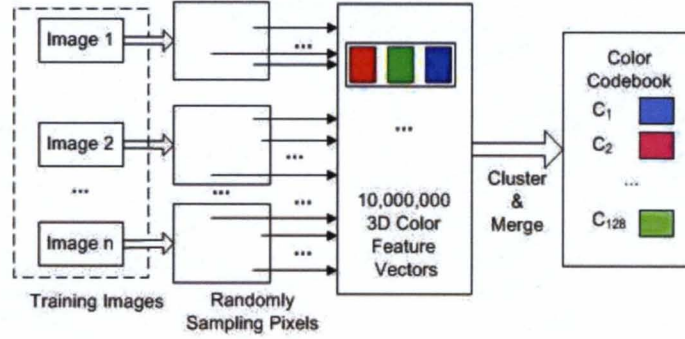


Figure 27: Process of Generating Image Color Codebook.

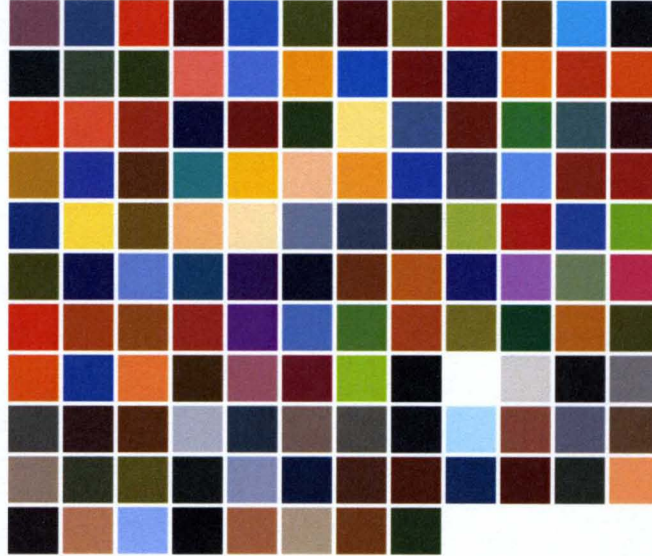


Figure 28: Example of a Color Codebook.

For our image color codebook design, our purpose is to generate a color codebook of size 128 (according to the sparseness distribution), which we denote as $C = \{C_1, C_2, \dots, C_{128}\}$, where each color $C_i = (r_i, g_i, b_i)$ is a 3-dimensional RGB color vector. We choose 1000 images for training. Since each image contains approximately 150,000 (the typical image size is 500 x 300) pixels, it is computationally prohibitive to use GLA algorithm on such a big number of pixels (150,000,000), therefore we resort to a pre-sampling step. In the pre-sampling step, we randomly sample 10,000 pixels from each image, thus with 1000 training images

we get a total of 10,000,000 points for GLA clustering. After we use GLA clustering method, we get 256 clusters, which are merged to get 128 centers (codebook) for the colorwords.

As shown in Figure 27, we apply the Generalized Lloyd Algorithm (GLA) followed by a merging algorithm to obtain the final 128 global color codebook to be used in indexing and in querying. Figure 28 shows an example color codebook generated by the above method.

Two things are needed to apply the GLA methods to our problem: the initialization method and the distortion measure. For initialization, we choose random cluster centers. For distortion measure, we use the Euclidean Distance measure.

$$D_i = \sum_n \|x(n) - c_i\|^2, \quad x(n) \in C_i \quad (16)$$

where c_i is the centroid of cluster C_i , and $x(n)$ is the color vector at pixel n .

3.2 Extracting and Indexing Image Color Features

After getting the global color codebook, we can use it to extract the color features from images for indexing. In the process of training to compute the image color codebook, we need not pay too much attention to computational efficiency. However, for extracting the image color features, we need to pay attention to the performance because we don't want this task to slow down the crawling stage.

Unlike the method used in [40], in our method, we are going to find the nearest neighbor in the codebook for any given color vector, using a kd-tree structure for fast nearest neighbors computation. Using this method, we obtain the following image representation for each region.

$$f_c = \{(I_j, F_j) | I_j \in \{1, 2, \dots, 128\}, F_j = \text{Frequency of index entry } I_j; 1 \leq j \leq N\} \quad (17)$$

where I_j is an index into the color codebook C , F_j is the corresponding term frequency, and N is the total number of colors in the region.

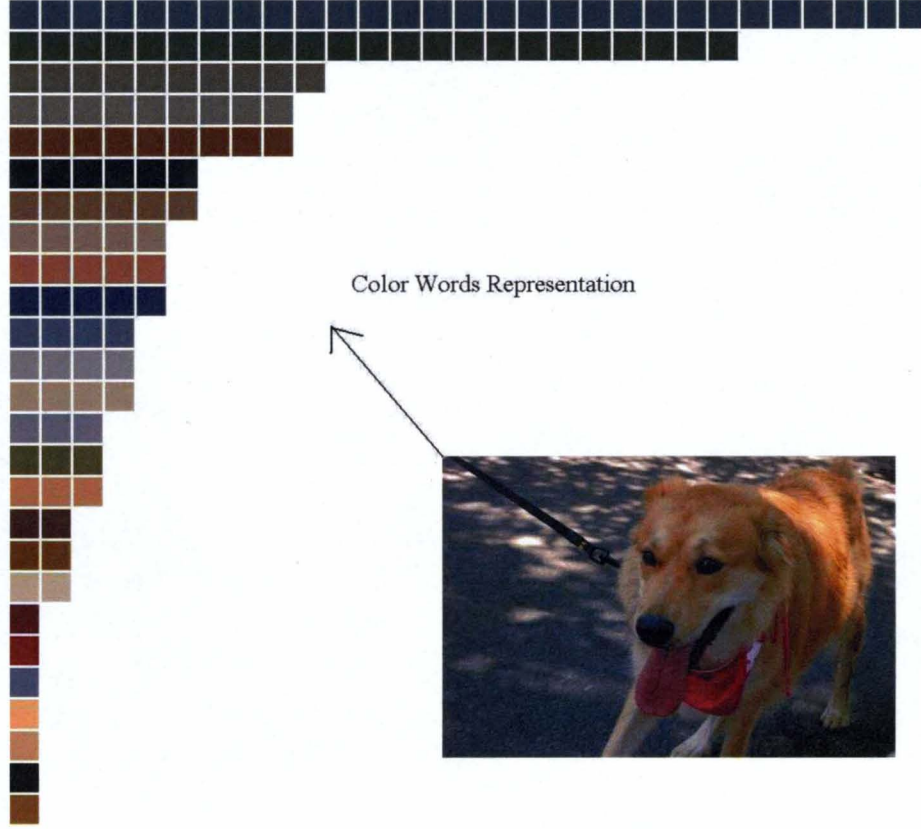


Figure 29: Extracting Colorwords from An Example Image.

When building an inverted index for these colors, we will scale these term frequencies F_j into appropriate indexable term frequencies by adopting a global scaling factor $T_{color} = 1024$. That is, if the occurrence of one color word falls below this threshold, we set the term frequency to zero and the corresponding index term I_j disappears from the representation. This way, we can build an inverted index by transforming these possibly very big numbers into corresponding indexable *Term Frequencies* used in text-like indexing. Therefore, Equation 17 is transformed into:

$$f_c = \{(I_j, T_j) | I_j \in \{1, 2, \dots, 128\}, T_j = F_j / T_{color} (\text{for } F_j \geq T_{color}) = \text{color term frequency}, 1 \leq j \leq N\} \quad (18)$$

Figure 29 gives an example of extracting colorwords from an example image. We can recognize the sparseness of the color words in this actual image. Only 26 out of the global 128 colorwords occurred in this image and among these 26 unique

colorwords, the two most popular colorwords occupy 36.9% of the total colorwords occurrences.

3.3 Image Tiling for Flexible Querying and Focused Retrieval

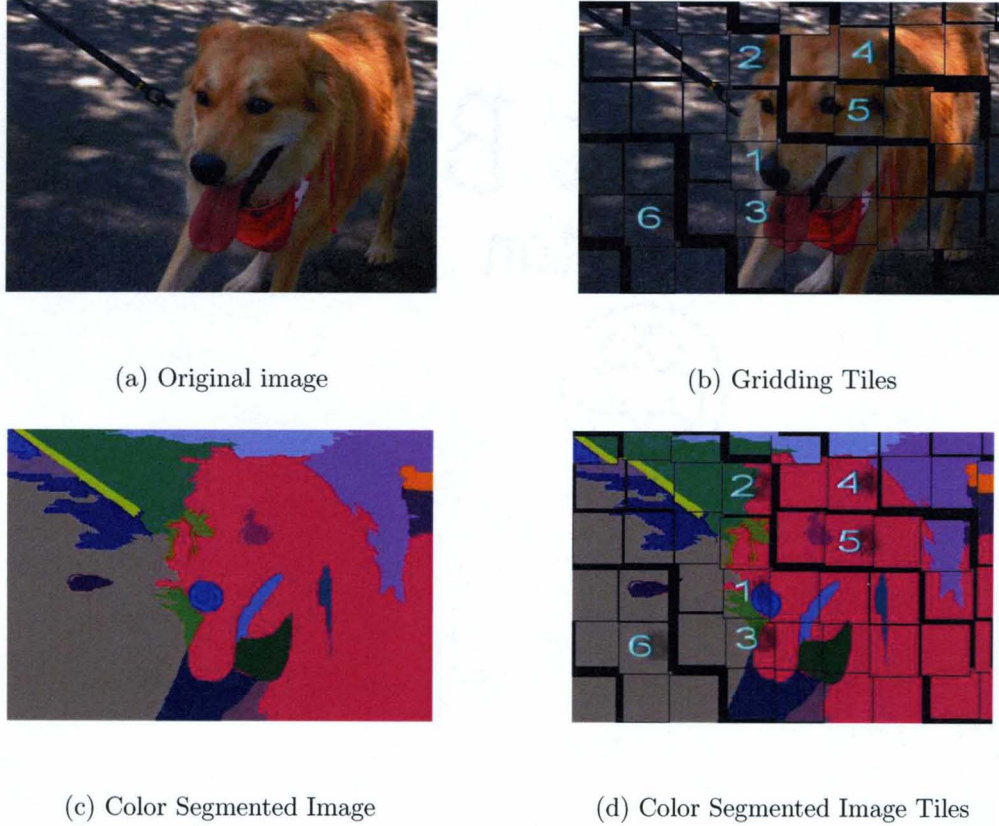


Figure 30: Different types of tiling in an Image.

Before we discuss texture words and shape words, we will define several relevant tile types that will support extracting such features.

Figure 30 shows how we divide an actual image from Flickr into small (64×64) tiles for further feature extraction. Notice that if we divide the original image directly into small grids (as in the top right image), some small tiles (tile 4, 5, and 6, etc) will have uniform texture patterns since they are totally inside one object or inside the background. We will name these tiles *inner tiles* as they lay totally inside an object or the background. On the other side, tiles such as tile 1, 2, and 3 don't have uniform texture patterns as they contain multiple conceptual areas

(the dog fir and the road). We name these tiles *bordering tiles* as they tend to border a big block and some other small blocks. *Inner tiles* are very helpful in identifying the texture pattern of an object, while *bordering tiles* are useful for identifying shapes or contours of objects.



(a) Original Image

(b) Color Segmented Tiles

Figure 31: Texture Image Without Large Segmentation Blocks.

Inner tiles and *bordering tiles* are not sufficient to characterize image tiles. For example, consider a texture image which may not have any large blocks or objects after segmentation. For instance, in Figure 31, when we segment and divide the brick wall image (left figure), we cannot get even one tile (right figure) which purely belongs to one big block or borders one big block. To deal with this case, we define a new another type of tile: *crossing tile*. By definition, it contains many small blocks without any pixel belonging to any big block. Notice that all the tiles in Figure 31 are *crossing tiles*. Such segmentation results often mean that the image is a texture image. *Crossing tiles* will be very meaningful in identifying texture patterns. For the above three types of tile cases, we will consider *bordering tiles* to be noise tiles for extracting *texture* features and thus ignore them when computing the Gabor texture feature. For shape analysis on the other hand, we will use *only bordering tiles*, while considering inner tiles and crossing tiles as noisy tiles.

model 1 Decision Tree Model for Classifying Tiles.

```
max <= 972
|   max <= 552
|   |   max <= 494: 2 (163.0/2.0)
|   |   max > 494
|   |   |   len <= 3: 1 (38.0/8.0)
|   |   |   len > 3: 2 (56.0/9.0)
|   |   max > 552
|   |   len <= 4: 1 (489.0/10.0)
|   |   len > 4
|   |   |   std <= 223.637
|   |   |   |   len <= 5
|   |   |   |   |   max <= 593: 2 (8.0/2.0)
|   |   |   |   |   max > 593: 1 (8.0/1.0)
|   |   |   |   |   len > 5
|   |   |   |   |   std <= 198.513: 2 (6.0)
|   |   |   |   |   std > 198.513
|   |   |   |   |   |   max <= 635: 1 (3.0)
|   |   |   |   |   |   max > 635: 2 (4.0)
|   |   |   |   |   |   std > 223.637: 1 (49.0/5.0)
max > 972: 0 (316.0/3.0)
```

In order to take advantage of tiling, we need to be able to decide whether an image tile is an inner tile, bordering tile, or crossing tile. To differentiate between these three kinds of tiles, we train a simple decision tree classifier (C4.5) to learn how to automatically classify a tile into the correct type. Note *inner* tiles contain one single segmented component based on color, *bordering* tiles would contain two or three color-segmented components of which there is one dominant component, while *crossing* tiles contain multiple color-segmented components neither of which is dominant. Hence, we use three features: number of components (*len*), number of

pixels of the maximum component (*max*), and standard deviation (*std*) of the number of pixels of each component in the segmented tile, and three class labels: inner tile (0), bordering tile (1), and crossing tile (2). We first use 1000 tiles as a training set. For these 1000 color segmented tiles, we manually label each segmented tile as inner tile, bordering tile, or crossing tile. After training and 10-fold cross-validation, we get the following classification model (with 93.8596 % Correctly Classified Instances).

3.4 Image Texture Codebook

We use Gabor texture feature extraction to generate the image texture codebook. Some related work includes [41], in which Ma et al used a two-stage training method to obtain an image texture thesaurus, and [42], in which Malik et al used the K-means clustering method to analyze the image texture using ‘*textons*’. Malik et al transform each pixel of an image into *texton channels*. We will adopt a similar method for texture extraction. However, because we first need to get the global image texture dictionary from a set of training texture images, instead of clustering the filtered image output *pixel-wise* for only one image as in [42], we will perform clustering *patch-wise* on an entire *set* of training images. This way, each texture image can be subdivided into small *patches* which contain certain *texture words*. We analyze texture patch-wise instead of textel-wise because this corresponds to our visual understanding of images: textures can make sense to us only when they are displayed in a *region* and thus form a pattern. However, if we study the filtered output with different orientations and scales, we may notice that these filters are for specific spatial points and not for patches.

Furthermore, we do not use all the segmented components for texture analysis. Instead we use only the inner tiles and crossing tiles. This is because first of all, image segmentation is not very accurate especially in the bordering part separating two components or blocks. Another reason is that sampling inner tiles instead of all the segmented blocks can help reduce the computational cost and raises the possibility of saving even more computations by only computing the

texture features of those tiles that are not similar to any other analyzed tiles. For instance, before we compute one tile's texture feature, we can check whether most of its pixel values are similar to the previous neighboring tile in the same block by taking the pixel value difference. If we find that the tiles are very similar, then we can simply classify this tile into the same pattern as the previous tile and avoid computing its texture features. The third reason for using blocks instead of tiles is that it would otherwise be very hard to handle crossing tiles, which are also very important in identifying texture tiles.

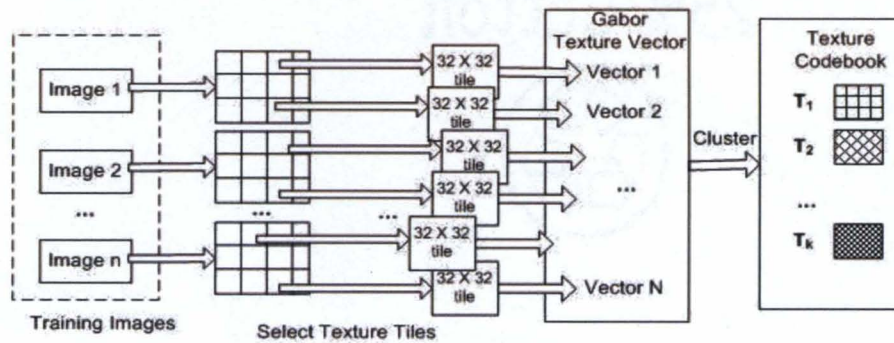


Figure 32: Process of Generating Image Texture Codebook.

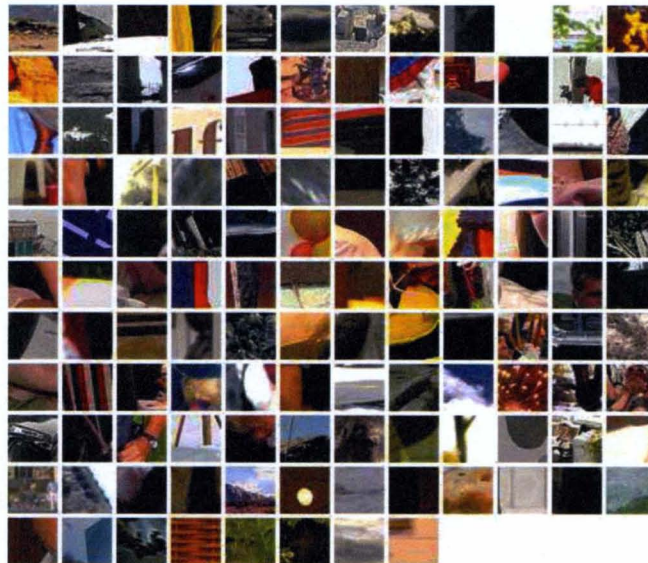


Figure 33: Example Texture Codebook.

To compute the texture codebook, we divide each image into 32×32 pixel blocks or tiles, select texture tiles (inner tiles and crossing tiles) according to image segmentation results, then apply Gabor filters to each texture tile, and finally record the corresponding texture features. The process for generating the image texture codebook is summarized in Figure 32. We use 1000 texture pictures for training (each picture roughly contains 160 (32×32 pixel) tiles with more than half of the tiles selected as texture tiles). This process yields 128 texture words (compared to [41] which yields 950 codewords). The clustering method used is GLA followed by the cluster-merge algorithm, as discussed above. Figure 33 shows an example texture book generated using the above procedure.

3.5 Extracting and Indexing Texture Features

As mentioned above, when we try to extract image texture patterns, we apply Gabor filters to several small 32×32 image patches or tiles which are selected as texture tiles. After extracting the image texture feature vector, we use Nearest Neighbor (accelerated by a Kd-tree structure) to determine the corresponding texture codeword from the texture codebook.

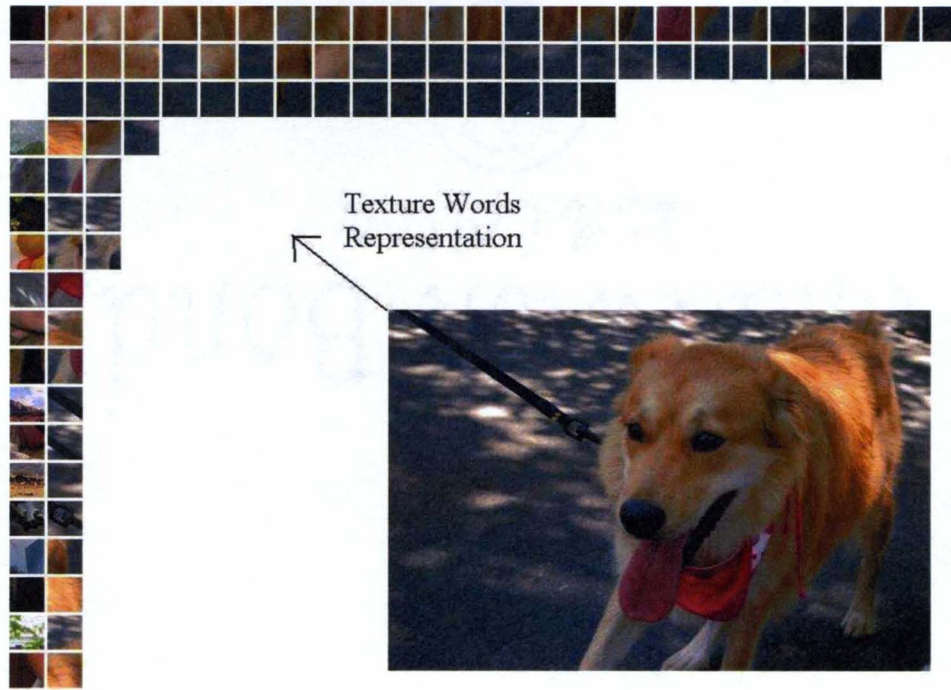


Figure 34: Extracting Texturewords from An Example Image.

Figure 34 gives an example of extracting texture words from an actual image. The leftmost column contains the representative tiles selected from the codebook in Figure 33, while the remaining tiles are the corresponding image tiles extracted from the actual image. In the process of extracting texturewords from an actual image, we adopt a similar procedure, where we segment the image and select texture tiles (inner tiles and crossing tiles) according to the segmentation results using the decision tree model show in model 1. We use a nearest neighbor search method to find the representative texture word in the texture codebook to represent each texture tile.

3.6 Training Set and Classifier Selection

The selection of a good representative training set is very important for the appropriate generation of the image codebook. For the world wide web image search case, we can probably never choose a training set that is large enough to cover every

possible pattern of color and texture. In other words, we can only use a partial set to represent the whole data set. All we can do is try to use a sufficiently large and diverse training set that contains as many patterns as possible. In the color codebook generation, we will try to select training images that contain as many visually different concepts as possible. For the texture codebook, we need to pay attention to diversity, that is, choose a training set that contains as many different texture patterns as possible. However, we also need to make sure that the “*correct*” texture patterns are extracted. To ensure this, first of all, we need to reduce the number of non-texture images in the training set. Second of all, we should try to choose images that contain uniform texture patterns. Last but not least, we need to guarantee that we can extract the uniform texture patches if the image as a whole is not uniform in texture. Thus, we need to use image segmentation in order to get uniform texture patches, as discussed above.

Finally, after generating the codebook, we will use a classifier for classifying the image content features to the corresponding codebook. The accuracy and efficiency of this classifier are crucial for the whole system to be effective and efficient. For color components, a straightforward nearest neighbor mapping maybe sufficient, however for texture tiles, a more sophisticated classifier may be needed to correctly decide whether an image contains a specific visual word or not. Unlike the text domain, where the occurrence of a term (word) in a specific document is categorical (hence requiring strict match), the occurrence of a visual word in an image relies more on the likeness of the image component to the corresponding codebook palette (hence requiring a more flexible and thus imperfect matching).

4 Indexing Image Tags and Content

4.1 Image Index Boosting

In section 2.2.3, we have discussed image “field boosting”. We discuss “document boosting” to images corresponding to different users. In chapter IV, we have also discussed how different image uploader profiles, image comments, etc,

could affect an image's ranking. For *image comments*, we currently only consider the number of comments posted for an image. Hence the boosting factor for image I is set as:

$$\beta_{image\ comments}(I) = \frac{N_c(I)}{k} \quad (19)$$

where $N_c(I)$ is the number of comments for image I , while k is a scaling factor (can be chosen as the average number of comments an image obtains). This means that the more comments an image has, the more popular it is, and consequently it gets a higher ranking. Note this boosting is in the indexing stage, and we also have a boosting policy in the querying stage.

Similarly, we will assign different *boosting factors* for images depending on their uploaders.

$$\beta_{image\ uploader}(I) = sim(T, P) \quad (20)$$

where T is the text in image's annotation, and P is the text in the image uploader profile. Basically, this means that if the annotations of an image page are similar to what the image uploader profile suggests, then that image would get a higher boosting factor.

4.2 Combined Indexing Scheme

After extracting the textual and visual information from the crawled images we integrate all of them for indexing. Figure 35 illustrates how the information obtained after acquiring the image color, texture, and shape content and the image *textual* annotations from the parent HTML file, is fed to *Nutch* (an open source Search Engine)'s built-in indexer (Lucene) for building an inverted searchable index.

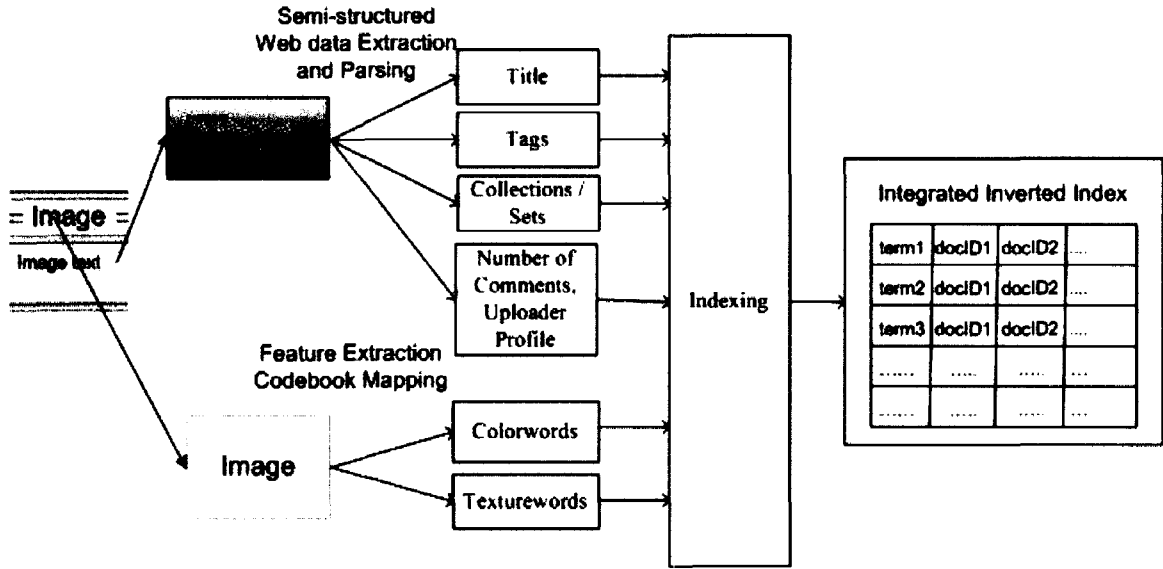


Figure 35: Indexing Images.

5 Summary and Discussion

In this chapter, we discussed image content indexing. Based on our review, we chose the inverted file structure for our indexing. We presented our *visual-word* generation methods for color, texture, and shape as image content features. For efficient and effective extraction of Gabor texture features, we defined three image tile types: *inner tile*, *bordering tile*, and *crossing tile*. Below, we summarize a few remaining open questions.

1. By converting image colors into textual words, do we lose some similarity information? For instance, will the fact that color “pink” is closer to “red” than “yellow” be lost? Even though we do lose such information, we can use query expansion as a remedy. For instance, when someone queries using the color “red”, we can expand the query to also contain “pink” but giving a lower boosting factor to “pink” compared to “red”.
2. How do we balance among different concepts that are expressed in a query? For instance, in the query “rose + color:red”, shall we pay more attention to the “textual” concept “rose” than the “image content” concept “color:red”?

We will address these problems in more detail In Chapter VI, when discussing the query interface and the weighting schemes design.

3. Do we lose the “spatial information” of the image contents after indexing? As in the case of *document* analysis and indexing, the spatial information (relative position of terms or contents) is also lost. In this case, we pay more attention to *Boolean Queries* than to accurate object queries for all the content features. Also, we could use a grid-based query as a remedy.

CHAPTER VI

Image Querying

1 Introduction

A search engine’s query interface is extremely important as it interacts with users directly. The purpose of the query interface and its functionalities is to make our image search engine accessible to most users without heavy learning. In chapter V, we addressed the problem of transforming image content features into text-like (codewords) for indexing. In this chapter, we discuss how we enable users to query the engine using these content features and using text (keywords). What makes our image search engine different from most other content-based image retrieval systems and commercial image search engines is our unique assortment of specialized query options. However, we need to ensure that these query functionalities will indeed help users get good quality results without causing unnecessary difficulties. The topics in this chapter include: a detailed discussion of each query functionality, the integration of different query functionalities, result ranking scheme, and finally query expansion techniques.

2 Query By Image Tags

As discussed in chapter IV, the final query ranking for the query by textual image tags (or keywords) will be based on the text in the image uploader’s profile and associated annotations, as well as the number of image comments. The image uploader profile indicates the authoritativeness of an image. The image’s associated text allows $TF \times IDF$ and field (such as title or tags)-based ranking mechanisms. The image comments indicate the popularity of the resulting image. The text-based

ranking of an image can therefore be expressed relative to a query Q using:

$$\begin{aligned}
 RankingScore(Q, I \in D) = & queryNorm(Q) \times \sum_{t \in Q \text{ and } t \in I/D} TF(t \text{ in } I/D) \times IDF(t) \\
 & \times boost(t.field \text{ in } I/D) \\
 & \times L^{-1}(t.field \text{ in } I/D) \\
 & \times \beta_q(t) \\
 & \times \beta_c(I) \\
 & \times \beta_p(I) \quad (21)
 \end{aligned}$$

- t : terms or words

- $t.field$ = field where term t is contained

- I/D an image or the document that contains the image, if a query is content query, it should be I , if the query is a text query, it should be D

- TF : term frequency. $TF = \sqrt{f_{I/D,t}}$

- IDF : inverse document frequency. $IDF = \log(1 + \frac{N_I}{f_t+1})$, where N_I = number of images

- field dependent boosting factor: title boosting factor = 2, tags and sets names boosting factor = 1

- $L^{-1}(t.field \text{ in } I/D) = \frac{1}{\sqrt{\text{number of terms in the field}}}$: Length normalization value of a field, given the number of terms within the field. This value is computed during indexing and stored in the index.

- $\beta_q(t) = IDF(t) \times (query \text{ boost})$: Boosting to one query term or one sub-query in case of Boolean queries.

- $queryNorm(Q)$: Normalization value for a query Q, given by the sum of the squared weights of each of the query terms.

- β_c, β_q : comments and profile boosting factors have been discussed in chapter V.

3 Query By Image Content

Since most current commercial image search engines allow users to query images by keywords, we will not discuss querying by keywords alone. Instead, we will discuss keyword-based querying later in the context of integration with querying by image content.

3.1 Query By Image Color

(a) Limitations of existing systems

In Chapter II, we reviewed several systems of query by image color. Although these query-by-color systems made a big breakthrough by combining colors and keywords for search, several problems remain unanswered.

First, a proper search result *ranking* mechanism does not seem to exist. Although the implementation details are not available for the above mentioned systems, our experiments with searching using different keywords and color palette combinations, indicate that most of them use color proportion for ranking. That is, if a picture contains a higher proportion of “blue” than “red”, then its ranking for “blue” would be higher than “red”. This ranking mechanism is rather intuitive, but a second look may make one think differently. For instance, given a picture that may contain a high proportion of “white” background (e.g., sky) compared to “red” objects (e.g., flower), when we submit the query “flower red”, the picture with “red” flower on a “white” background should be ranked higher than a “white” flower with mostly “red” background.

Second, a well-rounded integration of different types of queries is not yet used in these systems. In fact, most systems address only *a single facet* (tag, color, etc) of the whole problem. Even the few systems that combine query-by-color and query-by-keyword features do so by treating these features *independently*. Furthermore, no system can currently allow users to search by keywords, color, and texture altogether in one boolean query.

Finally, no system currently exploits “social” web type information (such as

the uploader profile) in crawling, indexing, and search.

Later, we will discuss how we try to overcome these problems using query weighting, and integration of different types of queries.

3.2 Query By Image Texture

There is currently no standalone texture retrieval system. Although a system by Ma et al [41], uses a texture thesaurus for browsing aerial photos, no active prototypes were found for our comparison. Most other systems that do use the texture feature use it as a “content” component, in which image similarity search is *hidden* to users. In our system, we have transformed the texture feature into *texture words* so that users can leverage texture for searching. Since we have trained a texture codebook and used it for indexing, the ability to search using these texturewords would naturally follow the indexing stage. To do this, we will build a texture palette similar to the color palette based on the generated codebook. Since presenting too many *small* texture thumbnails may not help users clearly discern the texture patterns, we will need to present bigger texture palette pitures for each pattern than for each color (to make the texture patterns discernable). Thus, we will present only few frequent patterns at the first level, and optionally present more patterns in a second level only, if users choose to see more.

4 Query Interface

TABLE 3
Query Functionality Comparison 2

System Name	Textual	Content				Example
	QBK	color	texture	shape	ROI	QBE
Google Image	✓					
Flickr Search	✓					
QBIC		✓	✓	✓		
SIMPLicity						✓
Retrievr		✓		“sketch”		✓
CIRES						✓
IKONA					✓	✓
Tiltomo	✓					✓
Riya	✓					✓
Fotolia	✓	✓				
iStockphoto	✓	✓				
yotophoto	✓	✓				
“Show and Tell”	✓	✓	✓			✓

Note: ROI Region of Interest;

As reviewed in section 7 of chapter II, there are many query functionalities provided by different systems as shown in table 3. But up to now, no system has attempted to provide query-by-texture features. Texture was used in many systems purely as a supporting factor for query-by-example-image. In our work, we are trying to provide *stand-alone* query-by-texture, as well as a mechanism to *combine* it together with other common query options.

Our current query interface contains query by keyword, color palette, and texture palette as shown in Figure 36.

SHOW and TELL

Searching with Image Content And Keywords

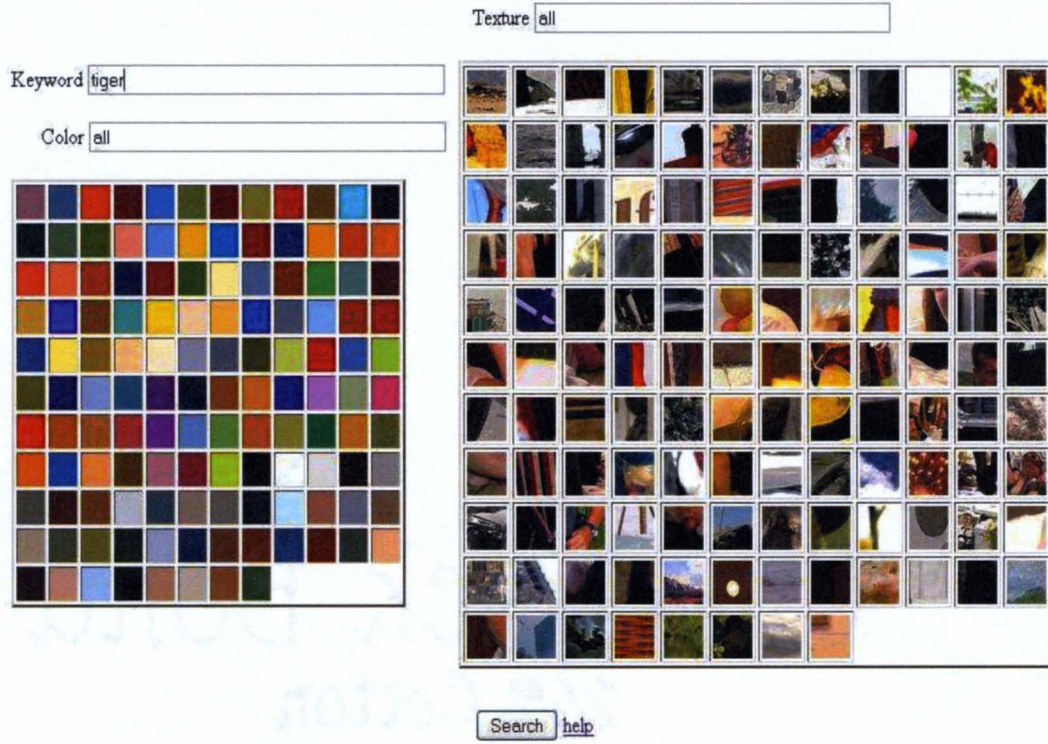


Figure 36: Current Query Interface.

5 Query Result Ranking

The BM25 [54] [53] ranking scheme is widely used as a state-of-art weighting scheme in text retrieval domain. We are going to adopt a similar measure, but we need to address the following concerns specifically for our image feature-based query, which is different from the above image-example-based query systems.

We have discussed using equation (21) for the final ranking, which is based on $TF \times IDF$. For our system to use the $TF \times IDF$ weighting, we need to normalize each feature space (tags, color, texture, boundary angle) to make them comparable, since we use different “textualization” methods for color, texture, and boundary angle features. Texture features and boundary angle features can be grouped together since they are both extracted through using image tiles and one

image tile can only be classified into either a texture tile or a boundary angle tile, and there is no overlap among them. For a typical image size of 500×350 pixels, that is divided into 32×32 pixel tiles, roughly 170 image tiles are generated. If we split these 170 feature words among texture words and boundary angle words, we would get $\frac{2}{3}$ of them as texture words because we have two type of tiles corresponding to the texture pattern and one type of tiles corresponding to the boundary angle pattern. This would be like a short essay containing only 170 words. Color is another facet of the image features. Each image tile may, in addition to containing texture, contain image color features, but we extract the image color feature globally (or regionally) instead of tile-based. Thus in the image color feature extraction process, we need to take care of the following two issues. First, we need to keep the ratio between the largest color component and the smallest color component to a reasonable degree, say less than 10 to 1, otherwise, we would risk overwhelming the query results with high color term frequencies given a boolean query containing both color and texture features. Second, we need to keep the total number of generated color words comparable to the number of texture words. We can also be more lenient toward the number of color words compared to texture words since color is more commonly accepted than other features. For example, we can limit the total number of color words (counting duplicates) to 170, which would then be close to the sum of the number of texture words. This will also help prevent the search results from being too biased toward colors in the boolean query scenario.

6 Query Expansion

Query expansion may be a bonus feature for text search, however it is indispensable for image content search. This is because human’s visual perception of objects is even more vague and imprecise than textual descriptions. For instance, some people may describe “pink” as being similar to “red” in certain scenarios. However, during indexing, we transform “pink” and “red” into different colorwords. Query expansion can provide one way to compensate for this mapping. When a user’s query contains the color “red”, we use Boolean “OR” to also include “pink”

(the expansion).

We use Euclidean Distance as a metric to help decide whether including the other term in the query expansion or not. We then use different boosting factors on each expanded term, which are inversely proportional to the distance from the query term to each expanded term. For instance, if the user query contains color C_1 such as “ $q = color : C_1$ ”, we find that we should include color C_2 and C_3 into the query expansion, if the distance values

$$D_2 = \|C_2 - C_1\| = \sqrt{(r_2 - r_1)^2 + (g_2 - g_1)^2 + (b_2 - b_1)^2} \text{ and}$$

$$D_3 = \|C_3 - C_1\| = \sqrt{(r_3 - r_1)^2 + (g_3 - g_1)^2 + (b_3 - b_1)^2}, \text{ respectively are below a given threshold. Then our query will be transformed}$$

into: “ $q' = C_1 + boost(\frac{1}{D_2}) : C_2 + boost(\frac{1}{D_3}) : C_3$ ”. The same technique is used for texture query expansion. The only difference is using their original vector dimensions in distance computation. For color, we have 3-dimensions (R,G,B). For texture, we have 48 dimensions or 60 dimensions depending on how many Gabor scales and orientations we use. Note that all these distance will be pre-computed off-line and will be available for use during the online query without adding any overhead.

7 Summary

In this chapter, we discussed different query functionalities for image search and made a comparison between our work and other works. Besides normal Query By Keyword and Query By Example, we provide query by image color and texture, which haven't been explored in other work. We have also discussed the query weighting scheme, and query expansion. All these methods strive to retrieve more relevant results for users' queries.

8 Open Problems and Discussion

It is true that we provide a more complicated image query interface, especially compared to Google's simple interface for example. However, we believe

that image search is inherently different from web page search. Web page search is generally biased toward popular pages, while in image search, people may be more interested in specificity than popularity. Similar to web page search, some people can take advantage of expressing their information need more richly by describing the desired image content.

Our multi-modal image search system differs from faceted image search although they share some similarities. Faceted search assumes that a user's interests flow along different facets (e.g. price, size, etc) as they browse or search, while our assumption is that users will persist in trying to find their target images by trying different filtering mechanisms. Our comprehensive functionalities provide the user the tool to do so. Integrating different features for ranking the final search results is never an easy task. Thus, we need to understand the users' real motive very well.

CHAPTER VII

Experimental Results

In this chapter, we first give the experimental results for each methods we've used, and finally give some snapshots of the current working prototype.

1 Demo Development Stages

1.1 Stage 1: Show and Tell (text and color)

In the first stage of our work, we developed a search engine that we named “show and tell” [71], and that used color words to describe image content and extracted URL text as the image textual annotations. There were several major limitations at this stage. For example, the content part was only restricted to the global image color histogram, and the textual part only contains URL text, etc. Yet the promising working results gave us motivation to improve the system. Query expansion was used only in stage one.

1.2 Stage 2: Text, Color, Texture

In the second stage of development, we expanded our image content part to include Gabor filter-based texture. We also crawled Flickr.com, to extract the image tags, and added tag clustering in the query-result-clustering part. The abundant tags in Flickr enriched our textual information, but also brought additional challenges on system efficiency and experimental validation.

1.3 Stage 3: Final Prototype

In order to overcome these problems, we have proposed our 3rd stage prototype design and implemented the current working demo. The major change

compared to the second stage is that we focuss less on the clustering of query results, and more on focused crawling. This stage also uses textual data extraction methods, introduces a visual codebook generation process, and introduce image segmentation in the content indexing part.

2 DOM Tree Path String Data Extraction

2.1 DRP Classification Experiment

As discussed in section 5 of Chapter III, we use a C4.5 decision tree classifier with 10 fold cross-validation. For the training set, we randomly downloaded 10 *DRPs* and 10 other types of pages from 5 web sites (Flickr, Youtube, ACM digital Library, SpringerLink, Amazon book), altogether we collected 100 pages with 50 *DRPs* and 50 other types of pages. We set the features as discussed in section 5 and obtained the following decision tree model.

```

fp-norm <= 0.036961: 0 (26.0)
fp-norm > 0.036961
|   max-sim<=0.144737: 0 (12.0)
|   max-sim > 0.144737
|   |   ps-ratio <=0.369231
|   |   |   max-sim <= 0.681818
|   |   |   |   max-sim <=0.184783: 0 (2.0)
|   |   |   |   max-sim > 0.184783
|   |   |   |   |   max-sim <= 0.277778: 1 (2.0)
|   |   |   |   |   max-sim > 0.277778
|   |   |   |   |   |   max-num <= 14: 0 (9.0)
|   |   |   |   |   |   max-num > 14: 1 (3.0/1.0)
|   |   |   |   |   |   max-sim > 0.681818: 1 (11.0)
|   |   |   |   |   |   ps-ratio > 0.369231: 1 (35.0)

```

Number of Leaves : 8

Correctly Classified Instances 85 85 %

We obtained a classifier accuracy of 85%. We then used this decision tree to classify 40 (20 DRPs and 20 other type of pages) new test pages randomly downloaded from Ebay and Buy.COM respectively. Altogether we got 80 pages for testing. The following table lists the results for each site respectively.

TABLE 4

DRP Classification Accuracy Result

	Ebay		Buy.com	
CLASS	DRP	NON	DRP	NON
DRP	14	6	15	5
NON	5	15	5	15
Accuracy	72.5%		75%	

We can see from the result that the overall accuracy was 73.75%. From this experimental result, we can see that although we have trained the classifier with one set of web sites and tested them with another two new websites, the classification accuracy is still high (around 73.75%). Thus, the heuristics that we chose seem to be website independent and to really capture the *DRP* patterns.

2.2 *DRP* Extraction Experiments

We have done a series of experiments for the simple scenario 1 on the Flickr image sharing site, and we found our approach very efficient and accurate for this specific web site case. The precision was very high and we could get all the accurate textual information. Hence in the following, we only give the experimental results for the more challenging scenario 2.

In this experiment, we compare our data extraction results with RoadRunner [13] and EXALG [4] on number of data values extracted. We used their web page examples and downloaded their experimental pages and their results from their web

sites ^{1 2}. For this experiment, we manually labeled the number of attributes and compared the number of attributes extracted from each work to the manually labelled one. The result is shown in Table 2.2, where N is the number of total pages for getting the schema, P is the real page No. that we compare, RR is the results from Road Runner, EXG is the results from EXALG, DPS is the number of values that our system obtained, and M is the number of manually labelled attributes.

TABLE 5

Data Extraction Comparison

Site	N	P	RR	EXG	DPS	M
Amazon(Car)	21	1	16	20	15	16
Amazon(Pop)	19	1	202	202	202	202
Buy.com(pro)	10	1	24	N/A	31	24
Buy.com(pro)	10	5	19/24	N/A	27	19
uefa(team)	20	1	9	9	9	9
uefa(play)	20	1	23/25	23	23	23
uefa(play)	20	5	22	22	22	22
MLB(play)	10	1	419	418	360	419
RPM(pack)	20	1	303	290	292	293

We can see from Table 2.2 that our results are comparable to existing results although our method is simpler and more intuitive. The processing time of our methods is $O(n)$, where n is the number of nodes in the *DOM* tree structure.

For a detailed comparison, in the Amazon car case, the extraction results of EXALG over-parses the page. For instance, it parsed the price range “\$29,030 - \$31,030” into two data values. The reason why we only have 15 attributes instead of 16 attributes is because every page contains at least one car, so the “1” was taken as the schema data. In the above data, we only took one specific page to compare.

¹<http://infolab.stanford.edu/~arvind/extract/>

²<http://www.dia.uniroma3.it/db/roadRunner/>

For Buy.com product information, RoadRunner has encountered the problem of disjunctions, it reported a couple of nulls for page 5 (5 missing data values), while our extraction method correctly extracted the data accordingly. We have more attributes than the manually labelled number. The reason for the out-numbered attributes is that for the data “Availability”, “Type” etc. info, although they occurred in every page, their *Path Strings* are different for pages with disjunctions and without disjunctions. So we extracted them all as data value instead of schema value. For MLB players, our method got much fewer data values than the manually labelled number, this is because we took the player’s position (“baseman”, “pitcher” etc) as the schema data as they occurred in every page in the same position in the DOM tree structure. This causes certain problems, but it can easily be addressed by marking the path string of that item as non-schema data.

2.3 Path String Differentiability Experiments

Another experiment that we have done for our extraction method is to see how the *Path String* can differentiate different schema data from real value data. Our assumption for using the *Path Sting* method to extract web data is that the path string for schema data and for real data share little in common. Thus, we can first use the path string to differentiate real data and schema data. And if the *Path String* can not totally differentiate among the two, then we can further use the node data value to differentiate them. Also we assume that using the *Path String* method, if we don’t need to consider schema *Path Strings*, then we save a lot of effort for extracting real data. For this experiment, we used “wget” to download the real web data from the popular sites, “Flickr”, “Youtube”, “Amazon”, etc. For each web site, we randomly downloaded 10 pages of the same type. For instance, in the Amazon book site, we only downloaded the pages that contain one detailed information of a specific book. For “Flickr”, we only downloaded the page that contains the detailed image page. We will name these pages *object pages*. After downloading these object pages, we *use* our implementation (written in java, and

using nekohtml parser ³ APIs for parsing the web page) to build the DOM tree and conduct our experiments.

TABLE 6

Path Strings Differentiability

Site	T	S	V	US	UV	INT
Flickr	133	111	22	36	16	3
Youtube	488	179	309	40	73	9
Amazon(book)	837	411	426	101	115	22
Ebay	474	183	291	56	113	15
SpringerLink	140	100	40	27	20	5
ACM DL	124	62	62	15	19	4

Table 6, T is the total *Path String Node Value* pair, S is the schema *Path String Node Value* pair, V is the value data *Path String Node Value* pair, and US is the number of *unique Path Strings* for schema data. Notice that some schema data with different text data value may share the same path strings. The same applies to value data. Different value data may also share the same *Path Strings*. UV is the number of *unique Path Strings* for Value data. Finally, INT is the number of intersections between US and UV. We can see from this table that our assumption is roughly satisfied. The small intersections between US and UV means that very few pages have the same *Path Strings* for schema data and for true value data. This tells us that we can indeed use *Path Strings* to differentiate between schema data and real data. Also, notice that the number of *unique Path Strings* is much lower than the number actual *Path String Node Value* pair (US is less than S, UV is less than V), this means converting from text node value *Path String* to *unique Path Strings* can save some time and space for processing.

³<http://people.apache.org/~andyc/neko/doc/html/>

3 Profile-based Focused Crawling

3.1 Topic Discovery Through Co-tagging

We tested two topics for the co-tagging topic discovery process using flickr photo-sharing site. In the first test, we give the starting tag “flowers”, and we collect 3,601 images whose tags contain the keyword flowers. From this 3,601 image tag sets, we find the following tags occur in the top co-tagging list (after removing a few noise such as “nikon”).

TABLE 7

top co-tagging tags for “flowers”

flowers	flower	nature	macro	spring
yellow	pink	garden	green	white
plants	red	flores	purple	blue

In the second round of test, we use the starting tag “nyc”, and after collecting 3,567 images whose tags contain “nyc”, we get the following expanded topic tag set.

TABLE 8

top co-tagging tags for “nyc”

nyc	new	york	city	manhattan
brooklyn	street	art	ny	newyork
graffiti	winter	park	gothamist	usa

Note that “macro” is a designation of the close-up kind of photography that is typically good to capture flowers. These results are meaningful to our knowledge. We then use these two sets of crawling topics for the following focused crawling experiments.

3.2 Profile Based Focused Crawling

The harvest ratio is often used to evaluate focused crawlers. It measures the rate at which relevant pages are acquired and how effectively irrelevant pages are

filtered. We will calculate the harvest ratio with the following formula

$$Harvest\ Ratio = \frac{N_r}{N_a} \quad (22)$$

where N_r is the number of relevant pages (belonging to the crawl topic) and N_a is the number of total pages crawled. To calculate the harvest ratio, we need a method to calculate the relevancy of the crawled pages. We use the following method. If the crawled page contains any of the tags that belong to the crawl topic, we would consider this page as relevant page, otherwise it will be considered as irrelevant. For comparison, we compared our focused crawling strategy with the breadth first crawling.

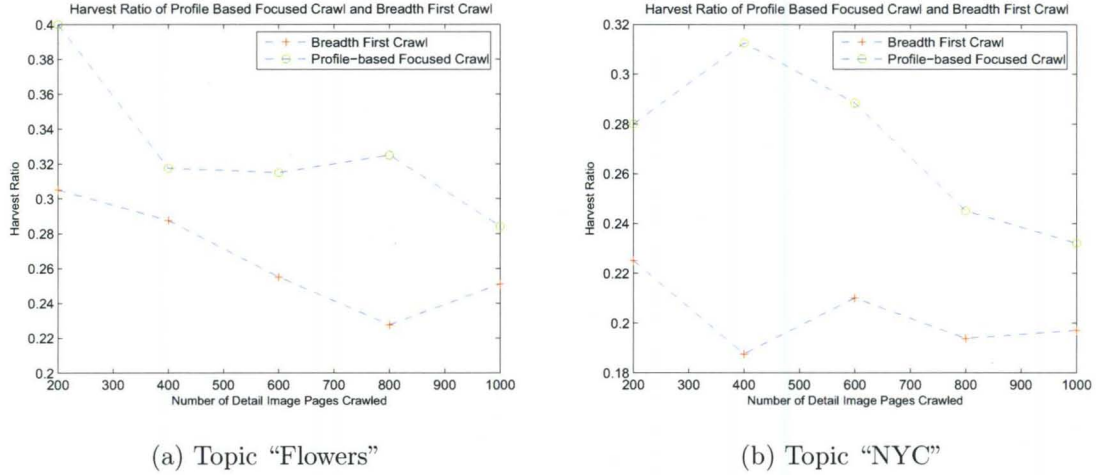


Figure 37: Crawling Harvest Ratio (threshold = 0.01).

We also conducted this test on the flickr photo-sharing site. We start our crawler with a list of urls with popular tags. Our first stage breadth-first crawler first records the uploader profiles that it extracted from corresponding pages. Later in our second stage of profile based focused crawling, we read these profiles, and calculate the corresponding ranks for each ourlink according to the user profile. In the focused crawling stage, we prune outlinks that are lower than a threshold value. Note that in the harvest ratio calculation, we only count the *detail image links* traversed. Figure 37 gives comparisons of focused crawling and breadth-first crawling for two crawling topics, "Flowers" and "NYC". We can see that our

harvest ratio for profile-based focused crawling is higher than that of the breadth first crawling.

In the following, we conducted groups of experiments to compare our profile-based focused crawler with that of the OPIC crawler for both the topic “NYC” and “Flower”.

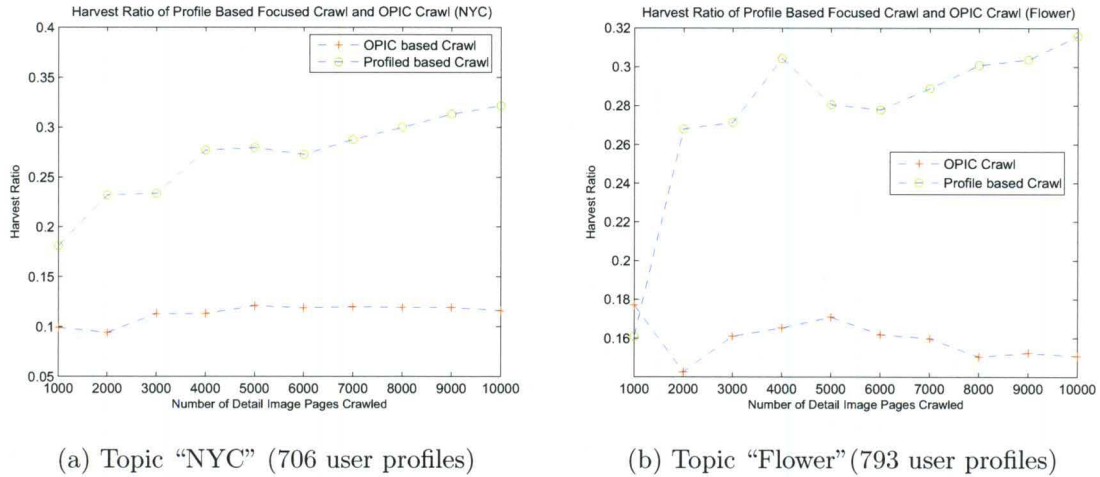


Figure 38: Crawling Harvest Ratio.

For the profile based crawler, we adjust the crawling strategy used by OPIC to take the user profile and crawling topic into account. Once we encounter a list page, if we find out that the list page is the crawling topic list by checking it’s URL, we will reset the score of that link to the initial maximum value (1.0). We reset the detail page scores or profile page link scores according to their corresponding user profile scores. For the rest of the links, we adopt the OPIC scores. We can see from the results that profile based focused crawling has a much better harvest ratio than purely OPIC based crawling. The experimental results for the harvest ratio are shown in Figure 38.

We also did experiments to compare the detail page capture ratio between profile-based focused crawling and OPIC based crawling.

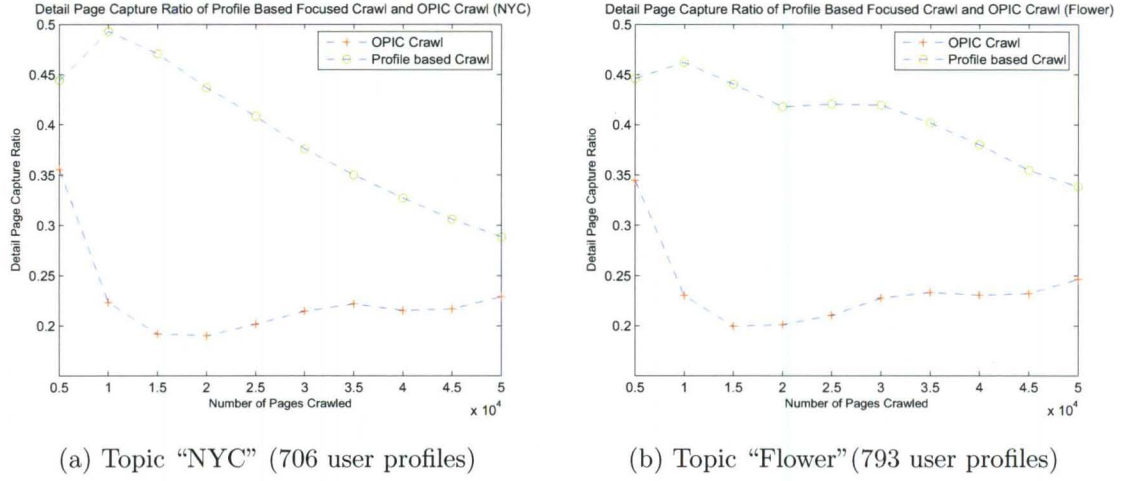


Figure 39: Crawling Detail Page Capture Ratio Comparison.

We can see that in both cases, the detail page capture ratio is higher for the profile based focused crawler than for the purely OPIC based crawler.

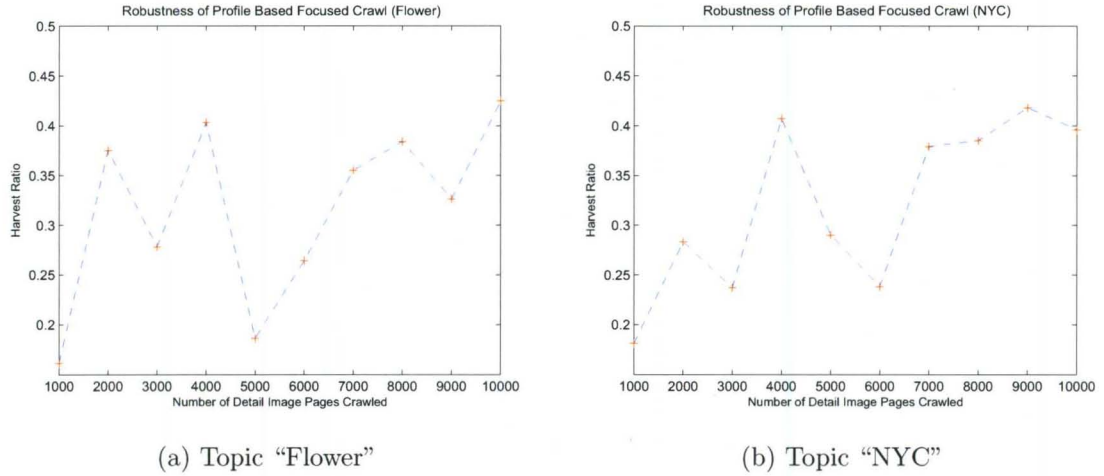


Figure 40: Robustness of Profile based crawler.

The robustness experiment serves to evaluate how stable a profile based focused crawler is. We did experiments on both topics with the following results. The difference between the robust experiments and the above harvest ratio experiments is that in the robustness experiments, we use a sliding window of 1000 pages to see the harvest ratio on each set of 1000 pages, while for the general harvest ratio experiments we measure the cumulative harvest ratio on 1000 pages,

2000 pages, ..., 10000 pages. From the experimental results, we can see that for both topics, the profile based focused crawler is robust.

4 Indexing Image Content

4.1 Effectiveness of Texture Word Representation and Comparison with CBIR

To demonstrate the effectiveness of our tile-based image representation, we will compare the retrieval precision of using image texture tiles with that of using the whole image texture vectors. Our assumption is that although we use a much faster boolean query consisting of several representative image texture words, we can get a comparable retrieval precision to that of using the whole image texture vectors for similarity search, which the general CBIR systems would use. To evaluate the precision, we adopt a similar strategy as was used in SIMPLicity [63]. We use a subset of COREL database with 10 categories shown in table 9, where each category contains 100 semantically coherent images. Altogether, there are 1000 images for testing.

TABLE 9
COREL Categories of Images Tested

Africa	Beach	Buildings	Buses	Dinosaurs
Horses	Flowers	Elephants	Food	Mountains

In the codebook generation stage, we collect 100 images by randomly selecting 10 images from each category. We use these 100 images to generate a global image texture codebook. After we build the image search by implementing Nutch⁴ on image texture feature, we conduct our testing by randomly selecting three images from each category as query images (30 queries for each case). To form the actual queries, for the texture words representation case, we select top-N (N ranges from 1 to 10) texture words for the query image to form the boolean query

⁴<http://lucene.apache.org/nutch/>

and we use Nutch's default $TF \times IDF$ ranking. For the texture vector query case, we take the whole 48 dimension texture vector for the query image to form the query and use Euclidean distance measure for ranking. We show 10 results in each page and examine the number of category matches in the first page. In case the total number of results is less than 10, we show all the results in the first page. We then calculate the precision as the number of category matches in the first page divided by the number of results in the same page. We then average the precision of the 30 result-sets and compare the two types of methods. The results is shown in Figure 41.

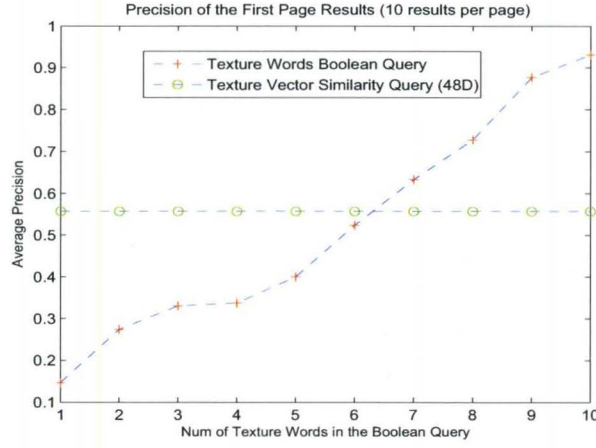


Figure 41: Texture Query Precision Comparison between Proposed Search and CBIR Search.

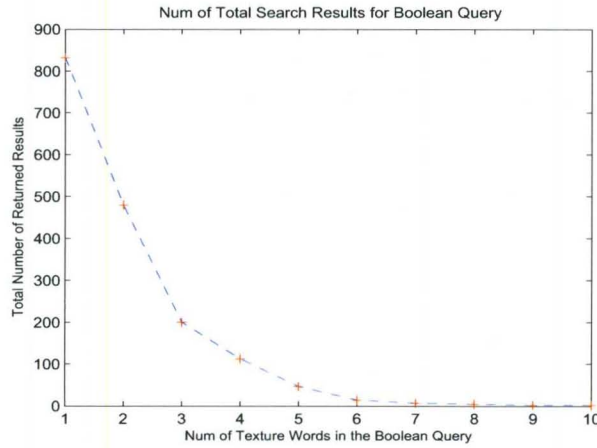


Figure 42: Number of Results Returned (Texture).

As shown in Figure 41, the precision for the Texture word boolean query case

increases as the number of query terms increases. As the number of query terms approaches 5 or 6, the average precision of the boolean query comes close to the vector similarity query case. On the other hand, the number of returned results drops dramatically as the number of query terms increases as shown in Figure 42. For instance, the average number of returned results drops from 13.5 to 6.6 as the number of terms in the query increases from 5 to 6. Although the average precision is high as we include much more terms (say 9 or 10), the very few number of results returned should prevent us from using too many terms. Combining Figure 41 and Figure 42, we can see that selecting around 5 query terms would give us a balanced result of desirable precision and total number of results.

On the other hand, the retrieval efficiency benefit of using the boolean query over vector similarity query is obvious, as shown in Figure 43. Here the experiments used a Linux server (with Intel(R) Xeon(TM) CPU 2.80GHz, 2cpu, and 2G memory).

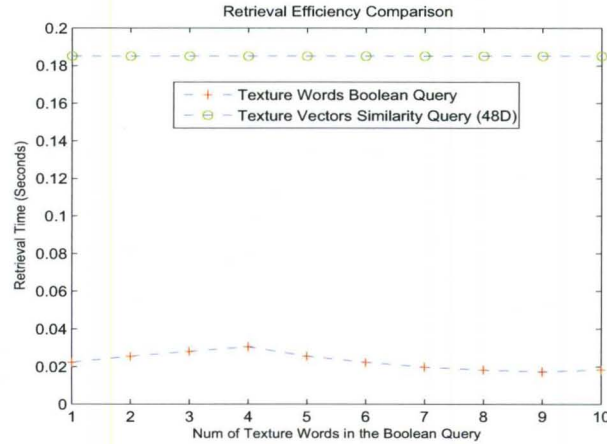


Figure 43: Texture Retrieval Efficiency Comparison between Proposed Search and CBIR Search.

We can also see that as the number of terms contained in the boolean query increases, the retrieval speed does not fluctuate much. This is guaranteed by the inverted indexing structure.

5 Current Working Prototype Evaluation

5.1 Deployment Configuration

We have two machines setup for the prototype, both with Linux operating system. In order to balance off the data, we use nutch distributed search functionality. On the webmining server, we run the demo portal, on another linux machine, we started a search server. We have four working interfaces for different evaluation purpose. The color palette and texture palette are displayed in Figure 28 and Figure 33 respectively in Chapter V.

TABLE 10
Demo Links

Application	URL	search server	Num of Images
US Travel Cities (focused)	http://webmining.spd.louisville.edu:8080/isearch/	webmining	16,056
breadth First Crawl	http://webmining.spd.louisville.edu:8080/bfs/	webmining	5,870
flowers (focused)	http://webmining.spd.louisville.edu:8080/flowers/	e-commerce	5,849
animal (focused)	http://webmining.spd.louisville.edu:8080/animal/	webmining	5,884

All the images were crawled and downloaded from flickr with a crawling depth of 6. In the following, we will give actual examples that compare the results of using a content query plus keyword query versus a keyword query only.

5.2 Snapshot 1: Content Color vs Keyword Color

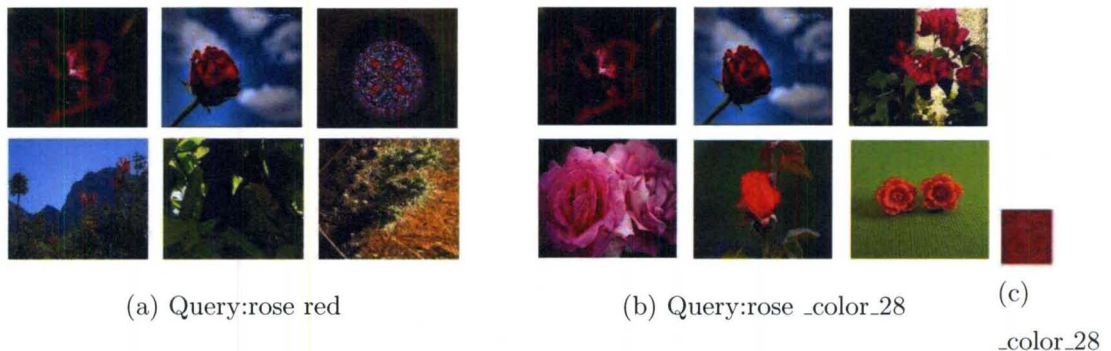


Figure 44: Query for Red Rose (from “flowers” demo link).

Figure 44 gives the snapshots of the result of querying “Red Rose”. Figure 44 (a) shows the top-6 results of using keyword only query. Figure 44 (b) shows the

top-6 results of using keyword plus image color palette as a query. The benefit of using the actual content color instead of the keyword color is obvious when our information need is color oriented.

5.3 Snapshot 2: Texture Words Filter Irrelevant Images

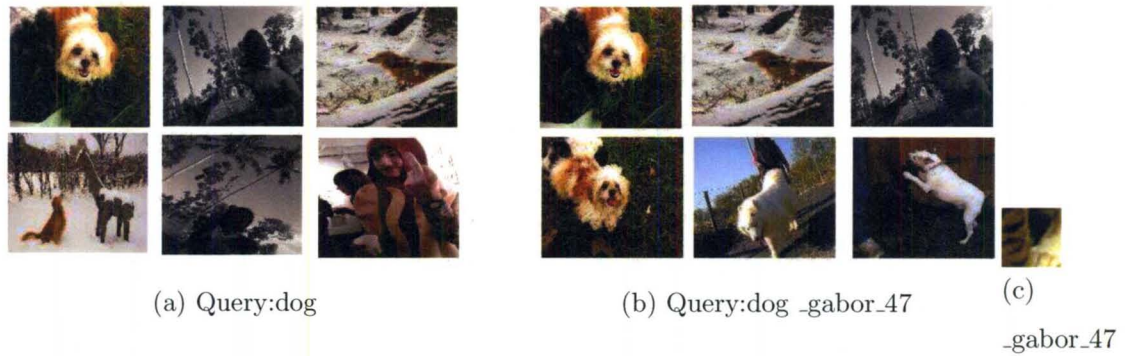


Figure 45: Query for Dog (from “flowers” demo link).

Figure 45 gives an example of using texture words to filter irrelevant images. The last image in Figure 45 (a) happened to be tagged “hot dog”, but is not really related to the actual dog. When we use texture words Figure 45 (b), we are able to filter out that image successfully.

5.4 Snapshot 3: Texture Words Help Find Target Images

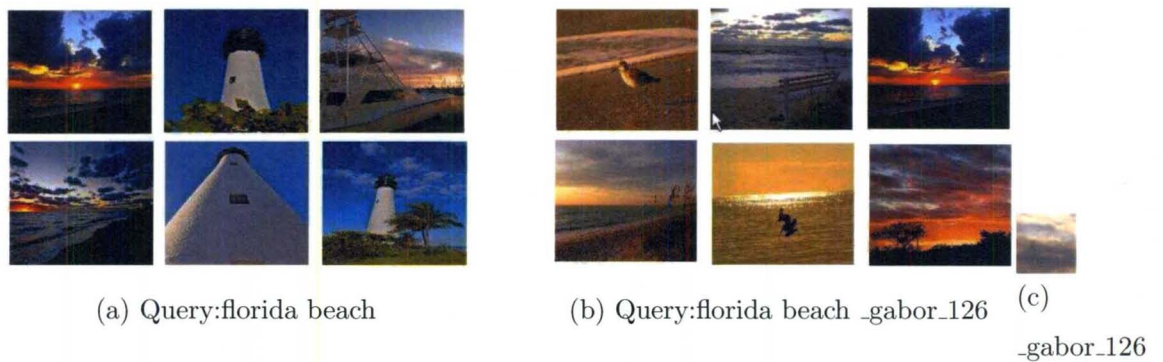


Figure 46: Query for Florida beach (from “flowers” demo link).

Figure 46 gives another example of using texture words to find relevant images. In this scenario, suppose that the user wants to find the picture of a real beach with water, sand, etc, but not their buildings. If they entered keyword query “Florida beach sand”, they would get no results since these tags were not used by the image annotators. This situation is frequent since the uploaders tend to use as few tags as possible to annotate the images. In this case, adding the texture content to the query was crucial to getting the right results.

5.5 Snapshot 4: Use “minus” option to filter irrelevant images

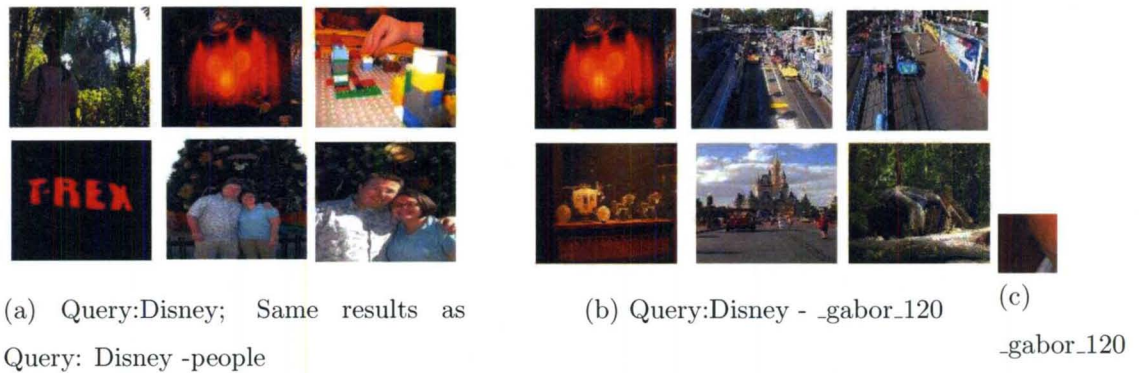


Figure 47: Query for Disney (from “US Travel Cities” demo link.

Figure 47 gives an example of using the “minus” option to filter out irrelevant images. We can use the “-” i.e. the “minus” or “not” option in the search box. Suppose that we don’t want to see people taking photos in the scenery, and use “-people”, this would still not filter the people out as shown in Figure 47 (a). While when we use “- texture-word” option and exclude the people texture of Figure 47, we can filter out the irrelevant images with people, as shown in Figure 47 (b).

6 Image Search Demo Evaluation

6.1 Search Functionality Evaluation of Second stage development

We have done a usability evaluation in the second stage. We invited 5 independent student evaluators from different majors with almost no image search

research and development background. We provided 100 queries, which are popular tags from *Flickr* ⁵. The evaluation items were 1) To compare image search using only keywords with using keyword plus color(texture). In our evaluation sheet, users can rate the search result as good (1) or bad (0). For example, users can input query “car red”, or input “car” and choose the red color from the color palette. They are requested to look at the results and report which one is better according to their judgement. 2) To see whether texture is helpful for search. 3) To see how boolean search: (keyword plus color plus texture) can help users filter unrelated images. Users can also use boolean “-” operation for “not”. For test 2 and test 3, users are requested to select “yes” or “no” to show whether the tested item was helpful.

TABLE 11
Users’ Evaluation of Search Functionality

Test Item	percentage
Keyword only good	12%
Keyword Plus Color Palette good	88%
texture helpful	70.6%
Boolean Query Helpful	61%

From the test results shown in Table 11, we can see that the keyword plus color palette query option gets higher ratings, while the texture palette was not as acceptable to general users as the color palette. This can be explained by the fact that human vision is more sensitive to color than to texture, which in turn makes the semantic gap between texture patterns and word annotations even wider than the gap between color and word annotations. This may also be due to the fact that texture tends to be harder to explain or “express” using common keywords. In general, however, the boolean query using color and texture seemed to help users in their image search.

⁵<http://www.flickr.com/photos/tags/>

7 Search Functionality Evaluation of 3rd stage development

In the 3rd stage development, instead of using a general crawl, we used our focused crawling strategy. We used flickr as our image tags and image annotation source. However, the images on flickr get updated with time. Therefore, to make the search results more comparable and have a stable source of analysis, we first used our data extraction process to extract all the images and their tags from flickr.com up to a depth of 6, starting with the most popular tags ⁶. Then we downloaded 23,692 images and saved the tags on our webmining server. Finally we ran a crawling and image indexing process in the webmining server.

7.1 Focused Crawling vs Breadth First Crawling (BFS) Comparison

Our purpose is to measure how focused crawling can give users more relevant search results compared to breadth-first crawling. We use two topics: “animal” and “flowers” and submit 10 queries for each. We ask the evaluators to compare the focused crawling result and the general breadth-first crawling results, by using the two links listed in Table 10.

TABLE 12
Profile based Focused Crawling vs Breadth First Crawling

Search Keyword	Total Results		Correct Results		Search Keyword	Total Results		Correct Results	
	bfs	animal	bfs	animal		bfs	flowers	bfs	flowers
cat	65	88	61	87	rose	10	62	7	26
dog	67	80	66	78	garden	58	86	57	83
pets	6	18	6	18	flora	4	24	4	24
zoo	77	162	63	99	tree	82	167	68	157
bird	62	111	56	106	spring	59	99	40	94
wildlife	13	17	12	16	green	72	246	67	180
frog	0	7	0	7	nature	160	420	160	417
nature	160	366	158	366	leaves	38	66	35	66
animals	63	67	54	62	park	221	295	172	281
tree	82	147	68	139	plants	11	34	11	33
total	595	1063	544	978	total	715	1499	621	1361

From Table 12, we can see that focused crawling gives about twice the number of results as the breadth first crawling, no matter how many total results or correct results. This confirms that if we have limited resources and want to get as

⁶<http://www.flickr.com/photos/tags/>

much information as possible for a specific topic, it is better to use our focused crawling method. To translate Table 12 into precision and recall, we would get the following Table 13, where BFRR is the breadth first crawl recall ratio.

$$Recall = \frac{NumofCorrectResults}{TrueTotalResults} \quad (23)$$

Since for both breadth first crawl and focused crawl, the “True Total Results” are the same, we have the breadth first crawl recall ratio as follows:

$$BFS-To-FocusRecallRatio = \frac{Recall(BFS)}{Recall(Focus)} = \frac{NumofCorrectResults(BFS)}{NumofCorrectResults(Focus)} \quad (24)$$

Table 13 summarizes the metrics of precision and recall ratio (BFS to focused), showing the superiority of our focused crawling strategy, since in all cases, the BFRR was less than 1 (on average around 0.5), meaning that more (on average twice as many) results are returned by focused crawling. Despite this larger recall, the precision has maintained its high average value (over 0.9).

TABLE 13
Precision and Recall Comparison

Keyword	precision (BFS)	precision (Focused)	BFRR	Keyword	precision (BFS)	precision (Focused)	BFRR
rose	0.7	0.419	0.269	cat	0.938	0.989	0.701
garden	0.98	0.965	0.687	dog	0.985	0.975	0.846
flora	1	1	0.167	pets	1	1	0.333
tree	0.829	0.940	0.433	zoo	0.818	0.611	0.636
spring	0.678	0.949	0.426	bird	0.903	0.955	0.528
green	0.93	0.732	0.372	wildlife	0.923	0.941	0.75
nature	1	0.993	0.384	frog	0	1	0
leaves	0.92	1	0.53	nature	0.987	1	0.432
park	0.778	0.953	0.61	animals	0.857	0.925	0.871
plants	1	0.97	0.333	tree	0.829	0.946	0.489
average	0.869	0.907	0.456	average	0.914	0.92	0.556

7.2 Search Functionality Evaluation

We also did a search functionality experiment for evaluating different search features. We crawled 16,056 images from flickr. We chose 10 popular travel cities (the first 10 in Table 14 from Yahoo travel’s top 10 US travel cities⁷. After our co-tagging topic expansion process, we obtained around 100 tags as the expanded

⁷<http://travel.yahoo.com/>

crawling topic, as shown in Table 14. It took around 20 hours to do our focused crawling on the US travel cities.

TABLE 14
Crawling Topic

nyc	chicago	seattle	sandiego	usa	orlando	boston
atlanta	california	sanfrancisco	losangeles	honolulu	florida	lasvegas
hawaii	newyork	oahu	newyorkcity	urban	city	april
downtown	china	manhattan	street	nevada	protest	architecture
tibet	red	spring	art	park	georgia	night
unitedstates	washington	water	sky	disney	beach	ny
canon	new	graffiti	blue	torch	building	vacation
flowers	york	video	olympic	birthday	people	olympictorch
flower	flag	baseball	massachusetts	waikiki	buildings	sign
travel	food	green	embarcadero	america	relay	brooklyn
sunset	skyline	lasvegas	nevada	usa	garden	film
universal	nature	olympics	clouds	waltdisneyworld	friends	wedding
december	ocean	music	black	disneyworld	trees	restaurant
flags	museum	bridge	white	tree		

For this evaluation, we gave 5 evaluators 30 images, shown in Figure 48 and Figure 49, randomly selected from the database. The purpose of this experiment was to find the target image from the search interface. For each image, we provided three hints to formulate queries: one hint on the keyword, one hint on the colorword and one hint on the textureword. Users could choose their own tags or they could use the query hints given to them. The tag hint was randomly selected from the image’s tags as annotated by the corresponding flickr uploader, while the color and texture hint were selected from the image’s processing results (after removing stopwords). We also asked the evaluator to record whether they could find the image using their own tags. Out of 30 target images, the evaluators reported an average of 20 images that they could find by themselves without using our query hints. That said, the evaluators reported that most tag hints were useful in formatting their queries.

The following description shows the questions asked to the evaluators about the quality of the query hints.

Query Hint Usefulness (0-3). (0 - useless, can find the image without hint;

1 - somewhat useful, need to go to next page to find the image; 2 - very useful, can find the target image in the first page; 3 - powerful, not only can find the target image in the first page, but all the images returned in the first page are strongly related.).

Query Hint Sensibility (-1 -2). (Does the hint match your own formulation about the image?) (-1- I think the hint is misleading, it does not match the image at all. 0 - not really, I would never use such a hint to search by myself; 1-probably, I would use this hint even by myself; 2-the hint matches the image perfectly, I would have used it even by myself)

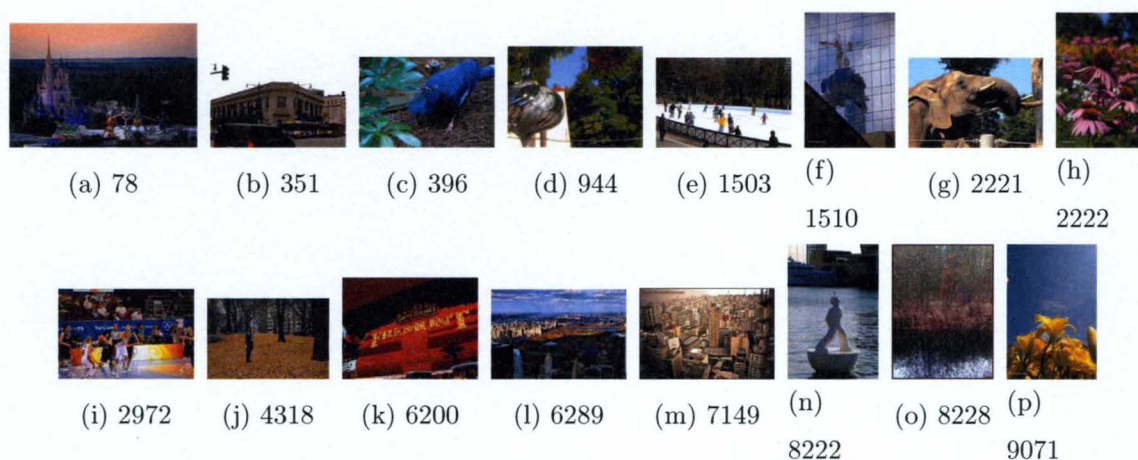


Figure 48: Evaluation Target Images Set 1.

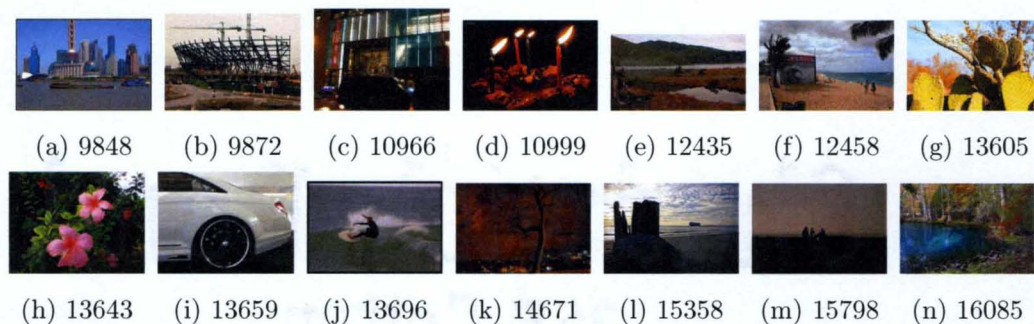


Figure 49: Evaluation Target Images Set 2.

TABLE 15
Search Comparison Between Tags and Tags Plus Contents

Image ID	Query Hints			Query Hint Usefulness in returning results						Query Hint Sensibility (i.e. score>0)		
	K	C	T	K	C	T	K+C	K+T	K+C+T	K	C	T
78	Disney	100	117	1	0	0	0	0	0	2	0	0
351	Chicago	92	99	1	0	0	1	1	2	2	0	0
396	Orlando	34	125	1	0	0	1	0	1	1	0	0
944	California	127	125	0	0	0	0	0	1	0	0	0
1503	Boston	92	69	1	0	0	2	2	2	2	1	0
1510	Chicago	53	117	0	0	0	0	0	2	1	0	1
2221	sandiego	122	48	0	0	0	2	0	2	0	1	1
2222	Flowers	111	77	1	0	0	0	0	0	1	1	1
2972	Beijing	97	59	1	0	0	2	2	2	1	1	1
4318	NYC	105	76	0	1	1	1	0	0	1	1	0
6200	vegas	94	123	0	0	0	2	2	2	1	1	1
6289	nyc	62	13	0	0	0	1	0	2	1	1	0
7149	building	41	125	0	0	0	2	0	2	1	1	0
8222	Barcelona	112	4	1	0	0	2	2	2	1	2	0
8228	Washington	0	117	0	0	0	1	0	2	0	1	0
9071	Summer	1	98	0	0	0	2	2	2	0	1	1
9848	China	16	34	0	0	0	2	2	2	1	1	1
9872	london	93	94	2	0	0	2	2	2	1	0	1
10966	MANHATTAN	97	26	0	0	0	1	2	2	1	0	1
10999	holiday	117	87	0	0	0	2	1	2	0	0	1
12435	tree	97	80	0	0	0	0	1	1	0	0	0
12458	sunset	99	24	0	0	0	0	1	2	0	0	0
13605	Nevada	104	62	0	0	0	2	2	2	0	0	0
13643	Hawaii	111	125	0	0	0	1	0	2	0	1	0
13659	los angeles	99	69	0	0	0	1	2	2	0	1	1
13696	surfing	100	2	1	0	0	2	2	2	2	1	1
14671	tree	97	6	0	0	0	0	0	0	0	0	0
15358	beach	99	80	0	0	0	0	0	0	0	0	0
15798	California	0	80	0	0	0	2	2	2	0	0	0
16085	park	83	125	0	0	0	1	1	1	2	1	0
average				0.333	0.033	0.033	1.167	0.967	1.533	0.733	0.567	0.4

From Table 15, we can see that the combination of tags, colors, and textures altogether can help users in finding relevant images more than each one of them alone, with an average satisfaction score of 1.533/3. Also, texture was the least helpful to the user. This is likely due to the much wider semantic gap between texture and the user’s own description about a target image. While some texture elements seem more intuitive, e.g. stripes, sand, clouds, waves, others are not so easy to understand or relate to a target concept in an image.

We have also received feedback from an evaluator who used our prototype for the first time and yet succeeded to obtain some target images without using any hints. The following are the specific results from this evaluator. She found 15 of the

30 images in the first page without using any hints in an average of 2-3 trials. This evaluator used “keyword” in 14 of 15 successful queries, used “color” in the query in 7 out of 15 successful results, and used “texture” along with color in 3 out of the 15 successful results. The evaluator also commented that a learning curve is needed (typically with 4-10 target images with given query hints). We expect that once this learning curve has stabilized, a user would experience more success and use the content words more often than this evaluator.

CHAPTER VIII

Conclusions and Future Work

In this chapter, we draw our conclusions and comment on our future work.

1 Conclusions

For image search, keywords can play a filtering role, while image content can play a semantic role in getting better results. Using only one of them in a query could lead to false positives. However, the *combination* of the image *keyword* and image *content together* in the same query, promise to reduce the occurrence of false positives as illustrated in many examples in chapter VII. We have described an image search framework benefiting from an integration of text and image content both in querying and indexing. Our framework also promises to inherit the scalability and performance of powerful text search engines.

In the DOM tree based DRP (Detail Record Page) extraction part, we have presented simple DOM tree *Path String* based methods for *DRP* identification and data extraction. We have adopted several heuristics to identify the *DRP* patterns from web pages based on record title and page title similarity and record *Feature Pair* occurrences. We also presented two scenarios for applying our *Path String* based method for real web record extraction. Our results and experiments confirmed their good performance. Our application to a flickr image set showed that text extraction is very efficient and simple.

In the focused crawling part, we have presented a profile based focused crawler, which ranks users who have more relevant images higher when we do the crawling. To further differentiate the component of a profile, we defined an *inner profile* to capture the user's individual topic interests and an *inter profile* to capture

the topic interests of the user's community of contacts. We have also used co-tagging to expand our crawling topic. In both the co-tagging topic discovery process and the profile-based focused crawling process, we used a *path string* based page classification scheme. The page classification was a required ingredient in our focused crawling because our ranking of links to crawl is computed based on profile, list, and information from detailed pages. Thus these needed to be distinguished from one another. Our experimental results confirmed the effectiveness of this classification method.

We have shown through our experiments that image content features follow a sparse power-law distribution. This further justified taking advantage of this sparseness for inverted file indexing. Our clustering-merge algorithm gave us better cluster center representations compared to GLA clustering, which, for sparse data, generated more cluster centers that were crowded in high density areas. Our experimental results showed our tile-based image *textualization* process for image texture and shape features are effective.

Based on all these methods, we have developed a working demo, which is available at <http://webmining.spd.louisville.edu:8080/isearch/>. It indexed around 16,000 images crawled and downloaded from flickr. Our evaluation shows that this search demo can help users obtain their target image more quickly, if they are trained to use the palette effectively. The integration of all these methods is suitable for social web media sites data extraction, crawling and search.

In order to do our evaluations, we have also collected and processed a benchmark data set that is available on <http://webmining.spd.louisville.edu:8090/test/raw/>. This data set can be used in the future by other researchers in this field without having to sacrifice excessive computational time and bandwidth to crawl and do the data extraction. The images and their annotations are already extracted.

2 Future Work

Bridging the user’s information search needs and the indexed data has always been a challenging task, especially when dealing with images and other multimedia data. Thus, an improved image search interface needs to be realized to make search easier, faster, and more accurate. However, the image shape based retrieval that we have outlined in our proposal can be included in future work. Finally, we mention that an online clustering process can greatly improve the quality of the indexed “content” as more images are added, without having to repeat the clustering for the entire index. One sensitive issue to take care of is that after updating the clustering results, the entire index may need to be rebuilt, using the “updated” cluster indices.

Some improvements to the final prototype (stage 3) are easy to do because they have already been implemented in the previous stage demos. These include: content query expansion, search by image example, and clustering of the results. We had to strip many of these refinements from our stage 3 prototype, in order to allow us to make fair evaluations, that can assess the basic mechanisms of focused crawling, content extraction and indexing, and search capability on their own, and without any interference from add-ons that will generally improve the results regardless of the underlying mechanisms. We anticipate that these additional features will be added to the social image search system in future stages.

REFERENCES

- [1] S. Abiteboul, M. Preda, and G. Cobena. Adaptive on-line page importance computation. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, pages 280–290, New York, NY, USA, 2003. ACM.
- [2] C. C. Aggarwal, F. Al-Garawi, and P. S. Yu. Intelligent crawling on the world wide web with arbitrary predicates. In *WWW '01: Proceedings of the 10th international conference on World Wide Web*, pages 96–105, New York, NY, USA, 2001.
- [3] C. C. Aggarwal, F. Al-Garawi, and P. S. Yu. On the design of a learning crawler for topical resource discovery. *ACM Trans. Inf. Syst.*, 19(3):286–309, 2001.
- [4] A. Arasu, H. Garcia-Molina, and S. University. Extracting structured data from web pages. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 337–348, New York, NY, USA, 2003. ACM Press.
- [5] N. Beckmann, H. P. Kriegel, R. Schneider, and B. Seeger. The r^* -tree: An efficient and robust access method for points and rectangles. In *SIGMOD Conference*, pages 322–331, 1990.
- [6] K. Bharat and G. A. Mihaila. When experts agree: using non-affiliated experts to rank popular topics. In *World Wide Web*, pages 597–602, 2001.
- [7] M. J. Black, G. Sapiro, D. Marimont, and D. Heeger. Robust anisotropic diffusion. *IEEE Trans. on Image Processing*, 7(3):421–432, 1998.
- [8] C. Carson, M. Thomas, S. Belongie, J. Hellerstein, and J. Malik. Blobworld: A system for region-based image indexing and retrieval. In *Proc. Third International Conf. Visual Information Systems*, pages 509–516, 1999.

- [9] S. Chakrabarti, B. Dom, and P. Indyk. Enhanced hypertext categorization using hyperlinks. In *SIGMOD '98: Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, pages 307–318, 1998.
- [10] S. Chakrabarti, M. van den Berg, and B. Dom. Focused crawling: a new approach to topic-specific Web resource discovery. *Computer Networks (Amsterdam, Netherlands: 1999)*, 31(11–16):1623–1640, 1999.
- [11] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *The VLDB Journal*, pages 426–435, 1997.
- [12] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, Cambridge, MA, USA, second edition, September 2001.
- [13] V. Crescenzi, G. Mecca, and P. Merialdo. Roadrunner: Towards automatic data extraction from large web sites. In *Proceedings of 27th International Conference on Very Large Data Bases*, pages 109–118, 2001.
- [14] C. Dagli and T. S. Huang. A framework for grid-based image retrieval. In *17th International Conference on Pattern Recognition (ICPR'04)*, pages 1021–1024, 2004.
- [15] M. Diligenti, F. Coetzee, S. Lawrence, C. L. Giles, and M. Gori. Focused crawling using context graphs. In *VLDB '00: Proceedings of the 26th International Conference on Very Large Data Bases*, pages 527–534, San Francisco, CA, USA, 2000.
- [16] E. A. El-Kwae and M. R. Kabuka. Efficient content-based indexing of large image databases. *ACM Transactions on Information Systems (TOIS)*, 18(2):171 – 210, 2000.
- [17] P. F. Felzenszwalb and D. P. Huttenlocher. Efficient graph-based image segmentation. *Int. J. Comput. Vision*, 59(2):167–181, 2004.

- [18] M. Ferecatu, N. Boujemaa, and M. Crucianu. Semantic interactive image retrieval combining visual and conceptual content description. *ACM Multimedia Systems Journal (to appear)*, 2007.
- [19] M. Flickner, H. Sawhney, W. Niblack, J. Ashley, and et al. Query by image and video content: the qbic system. *IEEE computer*, 28(9):23–32, Sept 1995.
- [20] C. Frankel, M. J. Swain, and V. Athitsos. Webseer: An image search engine for the world wide web. Technical report, Chicago, IL, USA, 1996.
- [21] A. Gersho and R. M. Gray. Kluwer Academic Publishers, 1992.
- [22] Z. Gong, L. H. U., and C. W. Cheang. Web image indexing by using associated texts. *Knowl. Inf. Syst.*, 10(2):243–264, 2006.
- [23] S. Grumbach and G. Mecca. In search of the lost schema. *Lecture Notes in Computer Science*, 1540:314–331, 1999.
- [24] S. Gupta, G. Kaiser, D. Neistadt, and P. Grimm. Dom-based content extraction of html documents. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, pages 207–214, New York, NY, USA, 2003. ACM.
- [25] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *ACM SIGMOD International Conference on Management of Data*, pages 47–57, 1984.
- [26] V. Harmandas, M. Sanderson, and M. D. Dunlop. Image retrieval by hypertext links. In *SIGIR '97: Proceedings of the 20th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 296–303, New York, NY, USA, 1997. ACM Press.
- [27] E. Hatcher and O. Gospodnetic. *Lucene in Action (In Action series)*. Manning Publications Co., Greenwich, CT, USA, 2004.
- [28] C.-C. Hsu and F. Wu. Topic-specific crawling on the web with the measurements of the relevancy context graph. *Inf. Syst.*, 31(4):232–246, 2006.

- [29] Y. Hu, G. Xin, R. Song, G. Hu, S. Shi, Y. Cao, and H. Li. Title extraction from bodies of html documents and its application to web page retrieval. In *SIGIR '05: Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 250–257, New York, NY, USA, 2005. ACM.
- [30] U. Irmak and T. Suel. Interactive wrapper generation with minimal user effort. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, pages 553–563, New York, NY, USA, 2006. ACM.
- [31] C. E. Jacobs, A. Finkelstein, and D. H. Salesin. Fast multiresolution image querying. *Computer Graphics*, 29(Annual Conference Series):277–286, 1995.
- [32] H. Jegou, H. Harzallah, and C. Schmid. A contextual dissimilarity measure for accurate and efficient image search. In *IEEE Conference on Computer Vision & Pattern Recognition*, June 2007.
- [33] F. Jing, M. Li, H. Zhang, and B. Zhang. An efficient and effective region-based image retrieval framework. *IEEE TRANSACTIONS ON IMAGE PROCESSING*, 13(5), May 2004.
- [34] F. Jing, M. Li, H. Zhang, and B. Zhang. A unified framework for image retrieval using keyword and visual features. *IEEE TRANSACTIONS ON IMAGE PROCESSING*, 14(7), July 2005.
- [35] F. Jurie and B. Triggs. Creating efficient codebooks for visual recognition. In *ICCV '05: Proceedings of the Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*, pages 604–610, Washington, DC, USA, 2005. IEEE Computer Society.
- [36] N. Katayama and S. Satoh. The sr-tree: an index structure for high-dimensional nearest neighbor queries. In *in Proc.ACM SIGMOD Conference*, pages 369–380, Tucson, AZ, 1997.

- [37] Z. Li, W. K. Ng, and A. Sun. Web data extraction based on structural similarity. *Knowl. Inf. Syst.*, 8(4):438–461, 2005.
- [38] B. Liu, R. Grossman, and Y. Zhai. Mining data records in web pages. In *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 601–606, New York, NY, USA, 2003. ACM.
- [39] H. Lu, B. C. Ooi, H. T. Shen, and X. Xue. Hierarchical indexing structure for efficient similarity search in video retrieval. *IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING*, 18(11):1544–1559, Nov 2006.
- [40] W. Ma, Y. Deng, and B. Manjunath. Tools for texture/color based search of images, 1997.
- [41] W.-Y. Ma and B. S. Manjunath. A texture thesaurus for browsing large aerial photographs. *Journal of the American Society for Information Science*, 49(7):633–48, May 1998.
- [42] J. Malik, S. Belongie, J. Shi, and T. K. Leung. Textons, contours and regions: Cue integration in image segmentation. In *ICCV (2)*, pages 918–925, 1999.
- [43] F. Menczer, G. Pant, and P. Srinivasan. Topical web crawlers: Evaluating adaptive algorithms. *ACM Trans. Inter. Tech.*, 4(4):378–419, 2004.
- [44] A. Mojsilovic, J. Kovacevic, J. Hu, R. J. Safranek, and K. Ganapathy. Matching and retrieval based on the vocabulary and grammar of color patterns. *IEEE Trans. on Image Processing*, 9(1):38–54, January 2000.
- [45] H. Mueller, D. M. Squire, W. Mueller, and T. Pun. Efficient access methods for content-based image retrieval with inverted files. In *Proc. SPIE Multimedia Storage and Archiving Systems IV*, volume 3846, pages 461–472, Aug. 1999.
- [46] A. Natsev, R. Rastogi, and K. Shim. Walrus: A similarity retrieval algorithm for image databases. *IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING*, 16(3):301–316, MARCH 2004.

- [47] Z. Nie, Y. Zhang, J.-R. Wen, and W.-Y. Ma. Object-level ranking: bringing order to web objects. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 567–574, 2005.
- [48] G. Pant and P. Srinivasan. Learning to crawl: Comparing classification schemes. *ACM Trans. Inf. Syst.*, 23(4):430–462, 2005.
- [49] P. Perona and J. Malik. Scale-space and edge detection using anisotropic diffusion. *IEEE Trans. Pattern Anal. Mach. Intell.*, 12(7):629–639, 1990.
- [50] B. G. Prasad, K. K. Biswas, and S. K. Gupta. Region-based image retrieval using integrated color, shape, and location index. *Comput. Vis. Image Underst.*, 94(1-3):193–233, 2004.
- [51] J. Qin, Y. Zhou, and M. Chau. Building domain-specific web collections for scientific digital libraries: a meta-search enhanced focused crawling method. In *JCDL '04: Proceedings of the 4th ACM/IEEE-CS joint conference on Digital libraries*, pages 135–141, 2004.
- [52] M. M. Rahman, B. C. Desai, and P. Bhattacharya. Visual keyword-based image retrieval using latent semantic indexing, correlation-enhanced similarity matching and query expansion in inverted index. In *10th International Database Engineering and Applications Symposium (IDEAS'06)*, pages 201–208, 2006.
- [53] S. Robertson, S. Walker, and M. Beaulieu. Okapi at trec-7: Automatic ad hoc, filtering, vlc and interactive track, 1998.
- [54] S. E. Robertson, S. Walker, M. Hancock-Beaulieu, A. Gull, and M. Lau. Okapi at TREC. In *In Proceedings of the Third Text REtrieval Conference*, pages 21–30, 1992.
- [55] H. T. Shen, B. C. Ooi, and K.-L. Tan. Giving meanings to www images. In *MULTIMEDIA '00: Proceedings of the eighth ACM international conference on Multimedia*, pages 39–47, New York, NY, USA, 2000. ACM Press.

- [56] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000.
- [57] K. Simon and G. Lausen. Viper: augmenting automatic information extraction with visual perceptions. In *CIKM '05: Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 381–388, New York, NY, USA, 2005. ACM.
- [58] J. Sivic and A. Zisserman. Video Google: A text retrieval approach to object matching in videos. In *Proceedings of the International Conference on Computer Vision*, volume 2, pages 1470–1477, Oct. 2003.
- [59] D. Squire, W. Muller, H. Muller, and J. Raki. Content-based query of image databases, inspirations from text retrieval: inverted files, frequency-based weights and relevance feedback, 1999.
- [60] S. Tong and E. Chang. Support vector machine active learning for image retrieval. In *MULTIMEDIA '01: Proceedings of the ninth ACM international conference on Multimedia*, pages 107–118, New York, NY, USA, 2001.
- [61] M. L. A. Vidal, A. S. da Silva, E. S. de Moura, and a. M. B. C. Joˆo. Structure-driven crawler generation by example. In *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 292–299, 2006.
- [62] J. Wang and F. H. Lochovsky. Data extraction and label assignment for web databases. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, pages 187–196, New York, NY, USA, 2003. ACM.
- [63] J. Z. Wang, J. Li, and G. Wiederhold. SIMPLIcity: Semantics-sensitive integrated matching for picture LIbraries. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(9):947–963, 2001.

- [64] J. Z. Wang, G. Wiederhold, O. Firschein, and S. X. Wei. Content-based image indexing and searching using daubechies' wavelets. *International Journal on Digital Libraries*, 1:311–328, 1997.
- [65] R. Weber, H.-J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proc. 24th Int. Conf. Very Large Data Bases, VLDB*, pages 194–205, 24–27 1998.
- [66] T. Westerveld. Image retrieval: Content versus context. In *Content-Based Multimedia Information Access, RIAO*, page 276C284, 2000.
- [67] I. H. Witten, A. Moffat, and T. C. Bell. *Managing gigabytes (2nd ed.): compressing and indexing documents and images*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999.
- [68] B. Wu, V. Goel, and B. D. Davison. Topical trustrank: using topicality to combat web spam. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, pages 63–72, 2006.
- [69] Y. Zhai and B. Liu. Web data extraction based on partial tree alignment. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 76–85, New York, NY, USA, 2005. ACM Press.
- [70] L. Zhang, L. Chen, F. Jing, K. Deng, and W.-Y. Ma. Enjoyphoto: a vertical image search engine for enjoying high-quality photos. In *MULTIMEDIA '06: Proceedings of the 14th annual ACM international conference on Multimedia*, pages 367–376, 2006.
- [71] Z. Zhang, C. Rojas, O. Nasraoui, and H. Frigui. Show and tell: A seamlessly integrated tool for searching with image content and text. In *In Proceedings of the ACM-SIGIR Open Source Information Retrieval workshop*, Aug 2006.
- [72] H. Zhao, W. Meng, Z. Wu, V. Raghavan, and C. Yu. Fully automatic wrapper generation for search engines. In *WWW '05: Proceedings of the 14th*

- international conference on World Wide Web*, pages 66–75, New York, NY, USA, 2005. ACM.
- [73] H. Zhao, W. Meng, and C. Yu. Automatic extraction of dynamic record sections from search engine result pages. In *VLDB '06: Proceedings of the 32nd international conference on Very large data bases*, pages 989–1000. VLDB Endowment, 2006.
 - [74] H. Zhao, W. Meng, and C. Yu. Mining templates from search result records of search engines. In *KDD '07: Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 884–893, New York, NY, USA, 2007. ACM.
 - [75] S. Zheng, R. Song, J.-R. Wen, and D. Wu. Joint optimization of wrapper generation and template detection. In *KDD '07: Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 894–902, New York, NY, USA, 2007. ACM.
 - [76] X. S. Zhou and T. S. Huang. Unifying keywords and visual contents in image retrieval. *IEEE MultiMedia*, 9(2):23–33, 2002.
 - [77] X. S. Zhou and T. S. Huang. Relevance feedback in image retrieval: A comprehensive review. *Multimedia Systems*, 8(6):536–544, April 2003.
 - [78] J. Zhu, Z. Nie, J.-R. Wen, B. Zhang, and W.-Y. Ma. Simultaneous record detection and attribute labeling in web data extraction. In *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 494–503, New York, NY, USA, 2006. ACM.
 - [79] Z. Zhuang, R. Wagle, and C. L. Giles. What’s there and what’s not?: focused crawling for missing documents in digital libraries. In *JCDL '05: Proceedings of the 5th ACM/IEEE-CS joint conference on Digital libraries*, pages 301–310, 2005.

- [80] J. Zobel, A. Moffat, and K. Ramamohanarao. Inverted files versus signature files for text indexing. *ACM Transactions on Database Systems*, 23(4):453–490, 1998.

CURRICULUM VITAE

NAME: Zhiyong Zhang

ADDRESS: Department of Computer Engineering and Computer Sciences
University of Louisville
Louisville, KY 40292

EDUCATION: M.S. Automatic Control
Beijing Institute of Technology
2002
B.S. Electrical Engineering
Jiaozuo Institute of Technology
1998

PREVIOUS
RESEARCH: Web Data Mining
Web Search
Information Retrieval

WORK: SINA, Beijing
2004-2005

China Aerospace Research Institute No.14, Beijing
2002-2004

PUBLICATIONS:
Z. Zhang and O. Nasraoui, "Profile-based focused Crawler for Social Media-Sharing

- Websites" accepted in EURASIP Journal for Video and Image Search, 2008
- Z. Zhang and O. Nasraoui, "Mining Search Engine Query Logs for Social Filtering-based Query Recommendation", Applied Soft ComputingC Special Issue on Dynamic Data Mining, Sep. 2008, 1326-1334.
- Z. Zhang and O. Nasraoui, "Feature Tile-based Image Visual Codewords Extraction for Efficient Indexing and Retrieval", International Conf. on Pattern Recognition (ICPR), Nov 2008.
- Z. Zhang and O. Nasraoui, "Profile-based focused Crawler for Social Media-Sharing Websites", IEEE International Conf. Tools with Artificial Intelligence (ICTAI), Tampa, FL, Nov 2008.
- Z. Zhang and O. Nasraoui, "Efficient Web Recommendations Based on Markov Clickstream Models and Implicit Search". In Proceedings of IEEE/WIC/ACM International Conference on Web Intelligence, San Jose, CA, 2007.
- Z. Zhang, C. Rojas, O. Nasraoui, and H. Frigui, "SHOW AND TELL: A Seamlessly Integrated Tool For Searching with Image Content And Text". In Proceedings of the ACM-SIGIR Open Source Information Retrieval workshop, Seattle, WA, Aug. 2006, 60-67.
- Z. Zhang and O. Nasraoui, "Mining Search Engine Query Logs for Query Recommendation". In Proceedings of the 15th World Wide Web conference, Edinburgh, UK, May. 2006, 1039-1040.
- Z. Zhang and O. Nasraoui, "Hybrid Query Session and Content-Based Recommendations for Enhanced Search". In Proceedings of the IEEE World Congress on Computational Intelligence, IEEE Conf. Fuzzy Systems, Vancouver, CA, June. 2006, 2068-2074.