Webからのオブジェクト関連情報の抽出

Object-related Information Extraction from Web

シエ シュアンソン* ジアン シャンシャン* ジェン ジチュアン*
Xuansong XIE Shanshan JIANG Jichuan ZHENG

要　旨

大量のウェブデータは個人利用にも企業利用にも価値のある情報源を提供する．ウェブページから情報を自動的に抽出することで，情報検索，意見マイニング，データ統計やトレンド分析などのいくつかのアプリケーションを大いに容易にすることができる．本稿では，単一ないし複数のオブジェクトについて説明するウェブページからオブジェクト関連の情報を抽出，統合するための方法を提案する．オブジェクトカテゴリ，オブジェクト識別子および属性値情報の3種類の情報が抽出される．エンティティ解決テクニックと属性名マッチングと値の正規化手法により，異なったウェブページからの同じオブジェクトは統合される．さまざまなウェブサイトからのウェブページにおける実験により我々の方法が有効であることを示した．抽出，統合の結果に基づくオブジェクト指向サーチエンジンを構築した．与えられたキーワードに対して，関連カテゴリ，オブジェクト，属性名および属性値を返すものである．表現形式として，単一オブジェクト，カテゴリ木または比較表が選択できる．初期の使用フィードバックによりサーチエンジンの有用性を確認した．

ABSTRACT

The tremendous amount of web data provides valuable sources for personal and corporate use. Automatically extracting information from web pages can greatly facilitate some applications such as information retrieval, opinion mining, data statistics and trend analysis. In this paper, we propose methods to extract object-related information (object category, object identifier and attribute-value pairs) from web pages describing single object or multiple objects. Then by using entity resolution, name matching, value normalization and clustering techniques, instances of same object from different pages are integrated. A preliminary object-oriented search engine is built based on the exprimental results from diverse web sites. Given a specific query, category tree and single object schema can be provided. Futhermore, comparable feature matrix can visualize relationship between objects and attributes. Initial usage feedback confirms the usefulness of the search engine.

* リコーソフトウェア研究所（北京）有限公司
Ricoh Software Research Center (Beijing) Co., Ltd.

1. Introduction

Web data mining and retrieval has attracted a lot of attention in the past decade from both the research and commercial worlds. As we know, most of the documents are un/semi-structured format. It's hard for machines to do statistics or further analysis based on those data. Furthermore, heterogeneous data widely exist; even for the same objects, there are different presentation ways at different sites. So how to convert un/semi- structured data into structured data and integrate them is our target.

In this paper, we consider a particular subset of web data which we call object-related web pages. They can be single-object pages or multi-object pages. Each single-object web page describes mainly a single object and lists explicitly some attribute-value information of the object. And multi-object web page lists several objects description and shows some comparable information.

There're a large number of object-related web pages such as product technical specification web pages from manufacturer, shopping, reviews and rankings web sites. Other examples include some encyclopedic web pages which describe information about an entity such as a country or even the Nobel Prize. These web pages contain rich sets of detailed object-related information including categories, identifiers, attribute names and values. Automatically extracting object-related information from those kinds of web pages can greatly facilitate some applications such as information retrieval, opinion mining, data statistics and trend analysis. Object information on multiple web pages can be aggregated to build object databases. Besides, the mined information can be useful for better indexing and searching in object-oriented search engines.

2. Related work

Generally, object-oriented search system faces not only web pages but also plain-text documents. In this paper, we focus on web pages information processing. Given an object-related web page, only three types of information, namely, object category, identifier and attribute-value information are extracted; other information, such as release information (time, region, and price) also can be extracted but this paper doesn't emphasize them.

Previous work on web object information extraction centers around product category and name extraction[1-5], class attribute extraction[6, 9], object attribute extraction with object and attribute resolution[10], and extraction from web tables[7, 8].

Observing those information extraction, for category, it's extracted from specific home pages[1] by template, or from plain-text[2] by cluster method; for identifier, some papers[3, 4] employ supervised machine learning method (such as SVM and CRF) by manual tagging and training. Prior art[5] shows how to find images to build text-based image search. The method in 10) extracts attribute text segments whereas ours focuses on extracting attribute names (with possibly multiple levels) and values simultaneously. The most related work[7] aims to extract semantic triplets (object identifier, attribute name, attribute value) from web pages.

Some differences from our work are as follows. For 1), it uses limited data source, and format of 2) are quite different with web pages. Methods of 3, 4) are time-consuming whereas our method is efficient, and the target of 5) isn't information extraction. In 7), it doesn't extract object categories which provide domain information about the object. And, 7) only extracts attribute name and value pairs while ours aims to extract attribute-value trees since attributes often have inherent hierarchical structures. A real-world example is the two-level attribute name tree (without values) including two paths, namely, "*lens system* → *type*" and "*lens system* → *optical zoom*". Without higher-level text "*lens system*", lower level attribute name "*type*" is ambiguous. Besides, "*lens system*" provides useful contextual information for lower-level attribute name "*optical zoom*".

3. Preprocessing

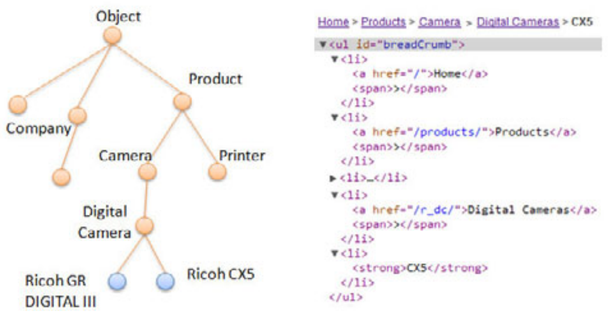
As precondition, the object-related pages should be collected and processed as input for following processes. The initial inputs can be seed categories or seed URLs from knowledge base, and more relevant URLs can be retrieved using existing search engines or topical crawlers after some query expansion methods. As precondition, those pages should be parsed into DOM trees by parser tool or web browser engine. Those tools parse the HTML code by analyzing its syntax, construct a DOM tree with computed style information, execute some dynamic content such as Javascript code, and render the web page on the browser window. The information we want to extract on a web page often exhibits visual patterns and the HTML code typically contains repetitive regularities. Therefore, the proposed algorithms heavily utilize the DOM tree and render visual information. Contents on multiple web pages are also used to complement each other. An advantage of these approaches is that they minimize the use of domain-related information.

4. Object-related information extraction

As above introduced, several useful information can be extracted. Generally, to build a basic object, we focused on 3 of them: category, identifier and attribute-values.

4-1 Category link extraction

The category link of an object is defined to be a sequence of terms denoting the hierarchical categories which are very useful for classifying different objects and building relationship among those objects; as an example, the ideal category link for the object, "*Ricoh CX5*" in Fig.1, is "*product* → *camera* → *digital camera*" which has 3 levels of category names.



Home > Products > Camera > Digital Cameras > CX5

```
▼<ul id="breadCrumb">
  ▼<li>
     <a href="/">Home</a>
     <span>></span>
   </li>
  ▼<li>
     <a href="/products/">Products</a>
     <span>></span>
   </li>
  ▶<li>…</li>
  ▼<li>
     <a href="/r_dc/">Digital Cameras</a>
     <span>></span>
   </li>
  ▼<li>
     <strong>CX5</strong>
   </li>
 </ul>
```

Fig.1 Category tree.

Category link is one of path in category tree. We assume a category link is a hierarchal link which comprises several levels, and the higher level shows wider concept which contains lower level. Basically, it's extracted from each web page based on DOM structural patterns, lexical features, position and other vision information.

(1) Filter-out impossible nodes by 3 factors: max depth from bottom to up, count of leaf children and specific stop words by:

$$Exist(node) = \{1 \le depth \le 2 \cap 2 \le count < 10 \cap \overline{stopwords}\}$$

Fig.2 shows the schematic diagram. Node A and B will be filtered; node C and D are candidates. After that, the candidate nodes of link are generated.

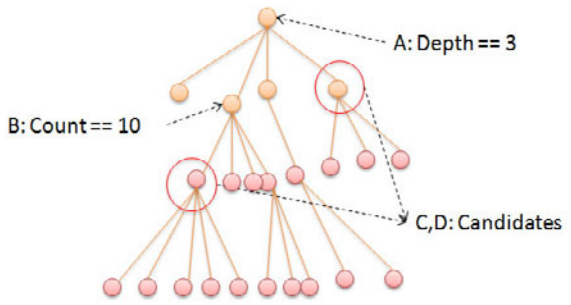


Fig.2 Example of category node filtering.

(2) Node type classification: convert all remaining nodes which have been filtered by the previous step (1) into 4 types, type A, B, C and D, by the following rules:

A. Node has value and NodeName == "A"

B. Node has value and NodeName == "TEXT"

C. Node has value and NodeName != "TEXT"

D. Node without value

(3) Pattern matching:

According to the steps (1) and (2), all the possible candidates are remained and we verify them by the following steps:

- Translate all leaves to a sequence with post-order traversal for each candidate; and each leaf has one of types, such as, A, B, C and D.
- Apply BIE (Begin, Interim, and End) patterns on above leaves based on node's type. Each pattern unit is identified by node's type; The Begin unit includes the first 2 leaves, such as AB, AC and so on; the End unit includes the last leaf, such as A, B; and the left leaves are taken as Interim unit which are repeat sequence by each 2 leaves, such as AB, BC and so on.
- Several BIE patterns with experimental weight are adapted to calculate the score of each candidate node. The experimental patterns are described as: B+I+E=weight. For example, AA+AA+C=0.9, AC+AC+C=0.8, AD+AD+D=0.3. Those patterns can be determined from expert's knowledge or statistics from some corpus.

Fig.3 shows the pattern matching processes.

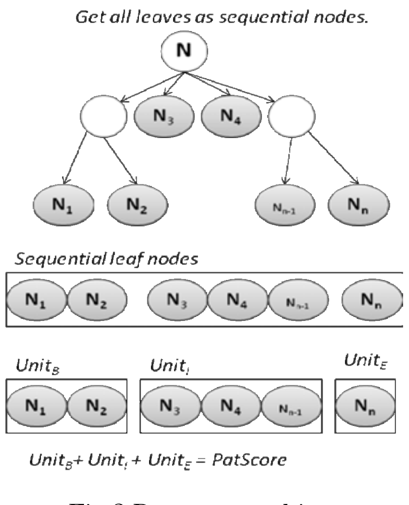


Fig.3 Pattern matching.

(4) Get vision score: Based on candidate nodes from the above steps, we'll calculate vision score for each node; for vision information, 2 factors are considered, one is absolute position, and the other is font.

- One of candidates for position calculation is 2D Gauss function, the formalism is:

  $H(u,v) = e^{-D(u,v)^2/2\sigma^2}$;

  $D(u,v) = \sqrt{(u-u_0)^2 + (v-v_0)^2}$;

  In that: u = PositionX, v = PositionY, u_0 = 180, v_0 = 180, σ = 200 for example. The constant number (u_0, v_0, σ) can be adjusted according to concrete tasks.

  The value of H is taken as position score of this node according to its position.
- The values of one dimension, X or Y, for all leaves are same, if not, it can be ignored. And the sequential leaves have increased position, if not, it can be ignored.
- The font of each leaf in a candidate node are same, if not, the candidate can be ignored.

(5) Merge above scores and get top 1 node, and it's taken as single hierarchical link after some symbols and unmeaning words removed. In this step, object identifier can be taken as additional weight: if one link includes identifier, then it's more like a taxonomy link.

(6) After got single hierarchal link from the previous steps, we can integrate multi links one by one into a directional graph and cut trivial edges based on nodes' frequency to build categories tree.

4-2 Object identifier extraction

A real world object always has a unique identifier (name) in common sense. We provide a method and procedure to find object names for specific topic from webpage, in which a name consists of several words, such as a product name, "*Ricoh CX3*". A topic, generally, means object categories, such as "*digital camera*". So, one embodiment is: find product names for a specific category.

The original categories are from section 4-1 and the candidate URLs are gotten from chapter 2. Due to different features of those 2 types, we adopted different approaches for finding identifier block from single-object and multi-object pages. After that, identifier fragment extraction, identifier refinement and ranking are applied. The overall processes are illustrated by Fig.4.

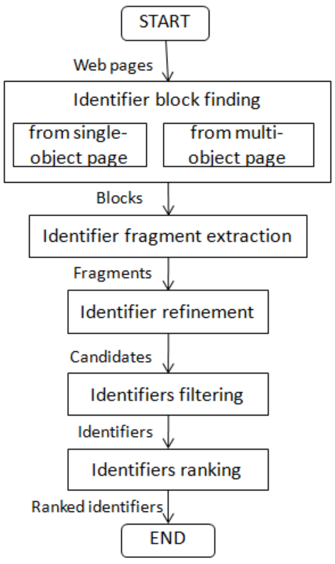


Fig.4 Identifier extraction processes.

4-2-1 Identifier block extraction from single-object page

This paper employs the following evidences to recognize identifier block:

(1) Vision information, similar to category's extraction, includes position, font, but with different parameters. Additionally, impossible nodes those in the same horizontal coordinate or vertical coordinate are removed. For font features, we adopt direct ratio with font-size, bold and so on. Structure information focus on header tag (such as "H1", "H2", "H3").

(2) Content information: Similarity with "TITLE" is one of factors. And specified regular expression may help to identify electronic product name. Furthermore, term frequency is calculated under an assumption that if a word more frequently occurred, it's more important and therefore the block is more important as an identifier.

Then those evidences bring different weights to DOM nodes. Identifier text could be picked out according to the sum weight. After all text nodes are weighted, the block with the highest score is determined as the object identifier block.

4-2-2 Identifier block extraction from multi-object pages

As we know, image is good for presenting a physical object or concrete issues. According to experience, those targets are widely described by image in multi-object pages. So we can take image node as important cue for names finding.

After got image nodes found by "IMG" tag, we get corresponding image names according to special attribute value of node, eg. "alt", "title", "src" and "href". If one of the image attribute-value exists, it should be checked by some patterns to determine whether it's a real image name or not. The checking factors include: string length, special symbols, and a predefined dictionary (eg. got from existing common English dictionary). After above steps, several image name candidates will be found with some noisy words, and will be filtered by matching nearby text elements in special sequences, which are generated by the following rules:

- ✓ All nodes will be assigned a type number according to its HTML tag name, eg. "TABLE" to 0, "TR" to 1, and "TD" to 2.
- ✓ Each node has a corresponding depth from root to leaf.
- ✓ Traverse the whole tree with pre-order, and record each node with number of type and depth.

Fig. 5 shows an example.

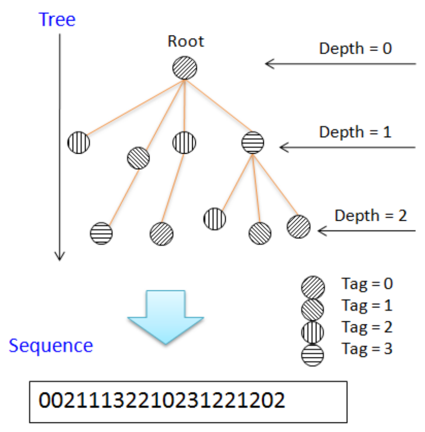


Fig.5 Translate tree into sequence.

After that, the name candidates are taken as inputs for repeated pattern finding. As we know, most of multi-object pages show comparable information, and the comparable identifiers are more authoritative compared with single name. By observing above sequence, we found: there are repeat sub-sequences for the comparable names in this sequence. We applied suffix array algorithm to find repeated sub-sequence.

Combined with above image names and sub-sequence weight by a threshold, the identifier blocks in multi-object pages will be found.

4-2-3 Identifier fragment extraction

Not all the results of identifier block recognition are perfect. Since the block is a text or an incomplete sentence, there could be noise which is irrelevant to object identifier. This step aims to remove noise and extract candidate identifier fragments which are supposed to be part of identifier unit. Given a sequence of words (identifier block), calculated by a weighted score, some words remain as identifier fragment. The weighted score calculation is based on the following features: located at the beginning of the block, not found in dictionary, and matching certain regular expressions.

Take "*Ricoh CX5 10 MP CMOS Digital Camera with 10.7x Optical Zoom (Pink)*" as an example. The feature-based scores are calculated, and then only words "*Ricoh*", "*CX5*" remain while others are removed as noise. Finally, the sequential words which are not separated by removed words are integrated into a unit as a fragment. In this case, only one unit is generated, "*Ricoh CX5*".

4-2-4 Identifier refinement and filtering

Actually, in most cases, identifiers extracted from single page are incomplete. Given identifier fragments, which are sequential units, the missing units could be repaired by multiple evidences from other pages.

For filtering, according to our experience, lots of names will be found for a category, but parts of them don't belong to the input category. We remove unmatched names as follows:

1) Filter by similarity: if any token of candidate can't be found in others, it's ignored.
2) Filter by topic: search results by exact name, and check it in title/snippet; if can't find any related information with topic, ignore it.

Some examples are illustrated in Fig.6.

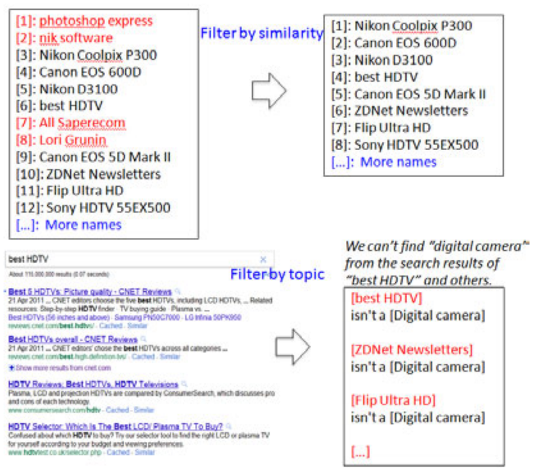


Fig.6 Example of identifier filtering.

4-3 Attribute-value extraction

To observe object-related pages, we found that single-object pages contain product specifications by some tables in which attribute-value pairs are listed, sometimes hierarchically. Therefore, we consider each single object table as an attribute-value tree. Our algorithm extracts attribute-value trees instead of just attribute-value pairs. In this tree, non-leaf nodes denote attribute names and leaf nodes denote attribute values. The algorithm mainly consists of two steps: construct a labeled node graph with scores; and select labeled node trees from the labeled node graph and then construct attribute-value trees. Each labeled node corresponds to a DOM node in the given web page and is associated with a label and a score. Each label indicates a classification of the DOM subtree rooted at the DOM node. A subset of labels includes: attribute, value, attr-val-pair, heading, attr-val-list, heading-attr-vallist where heading denotes a higher-level attribute name and attr-val is short for attribute-value. The score of a labeled node represents how probable the corresponding DOM node is of the corresponding label. The calculation of the labeled node graph is in a bottom-up matter where the score of the parent node depends on the children labeled nodes. In the second step, labeled nodes with highest scores are selected to generate the attribute-value trees.

Multi-object pages contain product comparison tables. In comparison table, a tuple which represents an object could be either a row or a column. We extract those relational tables by analyzing informativeness and coherence among potential object identifiers, attribute names and values. Firstly, raw tables are obtained from web page by selecting HTML nodes which are used to present tabular data, such as "TABLE", "TBODY", etc. Then contents contained in each table cell are normalized, because the content text may be null or there may be no text node but image nodes in a cell. After removing uninformative contents from tables, uninformative tables could be filtered. We analyze the candidate configurations of object identifiers, attribute names and values in table, and each candidate configuration is associated with a coherence score which indicates the possibility of this configuration. In each candidate configuration, the set of candidate object identifiers comprises a selected subset of rows or columns, the set of candidate attribute names also comprises a selected subset of rows or columns, and the set of candidate attribute values comprises the remaining table cells. The coherence score for each candidate configuration is calculated based on at least the following factors: whether the object identifiers are distinct, whether the attribute names are distinct, similarity scores for pairs of rows and pairs of columns, type consistency for each row and each column. Finally, candidate configurations with highest score for the table are selected. If the score is lower than a pre-specified parameter, this table is declared non-relational; otherwise, generate the resultant object identifiers, attribute names and attribute values according to this configuration.

5. Object information integration

Object-related information is extracted from certain web page; therefore we consider it as an object instance. Web pages represent information for different purpose, which leads to different wording, different style, and different point of views. It means, for a product object, there are hundreds of web pages talking about it, and there would be hundreds of distinct object instances. To build an integrated object database, an important task is to normalize the heterogeneous information, and provide a global view of the relationship between products and attribute-values in specific product domain. For category and name, we just applied similarity-based algorithm to identify whether there are same instances. In this chapter, we focus on attribute-values integration.

We employ a clustering-based method to normalize attributes, which includes attribute matching and attribute integration. Firstly, non-feature attributes should be cleared from object instances to construct an attribute pool. For example, if target product domain is digital camera and features are physical or functional, then remove the attributes which may vary from timeline (such as release information, review information, price information, etc.) by regular expression matching. Then we calculate pair-wise similarity between attributes, and result to a similarity matrix. We adopt a hybrid similarity matching method to ensure that attribute name, parent attribute name and values are all utilized. It means a pair-wise similarity score consists of name similarity score, value similarity score, and cross similarity score. Each kind of score has different weight. Name score measures distance between attribute names and parents names, by string similarity metric. Value score calculation depends on value normalization, for example, if a value includes numeral value and measure unit, it shouldn't be considered as a string. If the measure units are convertible, then compare the numeral values. We introduce cross similarity to explore more hiding relationships, such as "Pixels: 18000000" and "Resolution: 18 megapixels", the name score and value score couldn't connect them together, but if we measure the similarity between "Pixel" and "18 megapixels" which are attribute and value, we find them similar. The criteria for attribute matching is that, for attributes of same object similarity of values is more important; while for attributes of different objects, similarity mainly depends on attribute names.

To integrate attributes, we cluster them into features, by iterative clustering-ranking-filtering. The whole process is an optimization process which seeks to maximize or minimize a particular criterion function:

$$\frac{\sum_{r=1}^{k} n_r simScore[c_r][c]}{\sum_{r=1}^{k} \frac{1}{n_r} \sum_{i \in cluster(r)} simScore[i][c_r]}$$

Where, c is the center of all attributes, c_r is the center of cluster r, nr is the number of attributes in cluster r, simScore[x][y] is the similarity score of attribute x and attribute y. In which, the internal criterion function is to maximize cohesion in cluster (denominator), while the external criterion function is to minimize coupling cross cluster (numerator). In iteration, any clustering algorithm and optimizer can be employed to get an initial result, and we choose the centers as features, which represents similar attributes inside same cluster. Criteria for feature ranking is that, as a feature, the more attributes it represents, the more correct and important it is; position of a feature depends on positions of attributes it represents (here, position of a an attribute means its position in the instance):

$$Score_{feat(r)} = w_{sim} \sum_{i \in cluster(r)} simScore[i][c_r] + w_{position} \frac{n_r}{\sum_{i \in cluster(r)} i.position}$$

Then, there are three kinds of operations for filtering features. If two features are very similar, which means coupling score is more than certain threshold:

$$coupling(p,q) = \max(simScore[c_p][c_q], \frac{1}{2} \sum_{i \in cluster(p)} simScore[i][c_p] + \frac{1}{2} \sum_{j \in cluster(q)} simScore[j][c_q])$$

We merge the similar clusters into one, and re-calculate the center. If attributes a feature represents are not very similar, which means cohesion core is less than certain threshold:

$$cohesion(r) = \sum_{i \in cluster(r)} simScore[i][c_r]$$

We divide cluster into two clusters: pick two attributes which have minimum similarity as centers; distribute attributes to closest center. And if a feature represents very few attributes, which means a cluster has no attributes, or cohesion score of a cluster is less than certain threshold, we remove the cluster. After a minimum of the criterion function is achieved, we get the ranked features. Then feature matrix as one application is to determine the relationship between objects and features, which means fill the value cells in feature matrix. Additionally, normalized and integrated attributes could contribute to automatic domain ontology discovery and construction.

Table 1 Result of integrated objects.

	Product	Feature	Total cells	Null cells	Wrong	Correct	Accuracy (null: positive)	Accuracy (null: negative)	Accuracy (null: not count)
Digital camera	10	10	100	18	12	70	0.880	0.700	0.854
Smartphone	10	10	100	7	8	85	0.920	0.850	0.914
Printer	10	10	100	33	6	61	0.940	0.610	0.910
Projector	10	10	100	29	14	57	0.860	0.570	0.803
Ebook reader	10	10	100	49	13	38	0.870	0.380	0.745
Web browser	4	10	40	19	19	2	0.525	0.050	0.095
3D TV	10	10	100	30	11	59	0.890	0.590	0.843
3D cemara	4	10	40	14	10	16	0.750	0.400	0.615
Tablet PC	10	10	100	48	9	43	0.910	0.430	0.827
Hybrid car	10	10	100	10	9	81	0.910	0.810	0.900
Totally							0.846	0.539	0.751

6. Results

We collect web pages from different web sites with diverse formats and styles using some specific category, mainly, focused on electronics and new technologies related domain, and take 200 single-object pages and 400 multi-object pages to evaluate our algorithm. We select one web page from each web site since web pages from the same web site often have similar formats and styles, and answers of selected web pages are annotated manually.

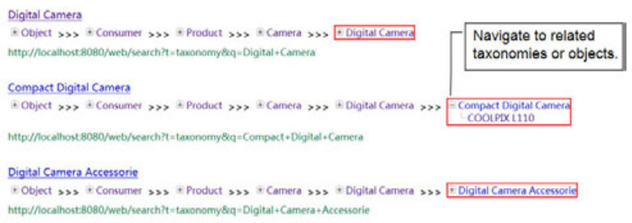
After extraction, we first calculate the F-measure value of the extraction results for each web page. Then the average F-measure value on all the collected web pages is reported. The F-measure results of object category extraction, object identifier extraction and attribute-value tree extraction are around 0.73, 0.82 and 0.78 respectively. And for object integration part, we evaluated our algorithm based on integrated matrix for top 10 products and features in different categories. The results are shown in Table 1.

Compared with existing method, our algorithms are web-site independent without template and training, and the results are acceptable in real applications. Further improvements are still probable using more refined patterns and cross-page confidence propagation methods.

7. Future work

After object database are built, some applications can be developed easily. An experimental application called "Object-oriented search" is one of examples. User can retrieve object-related information in category-level, object-level and attribute-level.

In category level, use can find all hierarchy categories which category/object belongs to. Fig.7 illustrates several results for "digital camera", which navigate to relative categories.


Digital Camera
Object >>> Consumer >>> Product >>> Camera >>> Digital Camera
http://localhost:8080/web/search?t=taxonomy&q=Digital+Camera

Compact Digital Camera
Object >>> Consumer >>> Product >>> Camera >>> Digital Camera >>> Compact Digital Camera
COOLPIX L110
http://localhost:8080/web/search?t=taxonomy&q=Compact+Digital+Camera

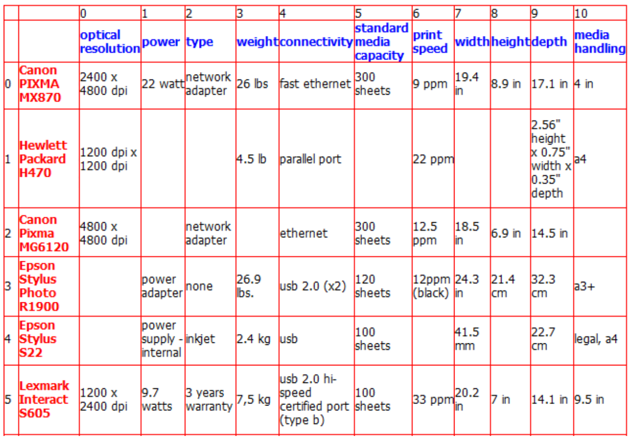
Digital Camera Accessorie
Object >>> Consumer >>> Product >>> Camera >>> Digital Camera >>> Digital Camera Accessorie
http://localhost:8080/web/search?t=taxonomy&q=Digital+Camera+Accessorie

Fig.7 Example of finding category tree.

In object-level search, user can find ranked objects in a category with attribute-values, and Fig.8 shows a matrix results for "printer" in this level, and those values are comparable and sortable. In many cases user wants to find objects with special attribute-value, and it can be implemented by some logical search, such as: Equal (=), More (>), Less (<), Similar (#).

Future applications include data statistics, trend analysis and data visualization. Obviously, we can get that information from structured object database using existing data mining methodologies.

Initial usage feedback confirms the usefulness of the search engine.



		0	1	2	3	4	5	6	7	8	9	10
		optical resolution	power	type	weight	connectivity	standard media capacity	print speed	width	height	depth	media handling
0	Canon PIXMA MX870	2400 x 4800 dpi	22 watt	network adapter	26 lbs	fast ethernet	300 sheets	9 ppm	19.4 in	8.9 in	17.1 in	4 in
1	Hewlett Packard H470	1200 dpi x 1200 dpi			4.5 lb	parallel port		22 ppm			2.56" height x 0.75" width x 0.35" depth	a4
2	Canon Pixma MG6120	4800 x 4800 dpi		network adapter		ethernet	300 sheets	12.5 ppm	18.5 in	6.9 in	14.5 in	
3	Epson Stylus Photo R1900		power adapter	none	26.9 lbs.	usb 2.0 (x2)	120 sheets	12ppm (black)	24.3 in	21.4 cm	32.3 cm	a3+
4	Epson Stylus S22		power supply - internal	inkjet	2.4 kg	usb	100 sheets		41.5 mm		22.7 cm	legal, a4
5	Lexmark Interact S605	1200 x 2400 dpi	9.7 watts	3 years warranty	7,5 kg	usb 2.0 hi-speed certified port (type b)	100 sheets	33 ppm	20.2 in	7 in	14.1 in	9.5 in

Fig.8 Example of finding comparable objects.

References

1) H. Davulcu, S. Koduri, and S. Nagarajan: A taxonomy based crawler for automated data extraction from data-intensive websites, WIDM, (2003).

2) V. Kashyap, C. Ramakrishnan, C. Thomas, and A. Sheth: TaxaMiner: an experimentation framework for automated taxonomy bootstrapping, International Journal of Web and Grid Services, (2005).

3) Y. Xue and Y. Hu: Web page title extraction and its application, Information Processing & Management, Vol.43, No.5. (September 2007), pp.1332-1347.

4) J. Zhu and Z. Nie: Simultaneous record detection and attribute labelling in web data extraction, KDD, (2006).

5) C. Frankel and M. J. Swain: WebSeer: an image search engine for the world wide web, The Univ. of Chicago, (1997).

6) E. Alfonseca, M. Pasca, and E. Robledo-Arnuncio: Acquisition of instance attributes via labeled and related instances, SIGIR, (2010), pp. 58-65.

7) E. Crestan and P. Pantel: Web-scale knowledge extraction from semi-structured tables, WWW, (2010), pp. 1081-1082.

8) W. Gatterbauer, P. Bohunsky, M. Herzog, B. Kr?pl, and B. Pollak: Towards domain-independent information extraction from web tables, WWW, (2007), pp. 71-80.

9) S. Ravi and M. Pasca: Using structured text for large-scale attribute extraction, CIKM, (2008), pp. 1183-1192.

10) T.-L. Wong, T.-S. Wong, and W. Lam: An unsupervised approach for product record normalization across different web sites, AAAI, (2008), pp. 1249-1254.