Semi-automatic Data Extraction from Tables

© Nikita Astrakhantsev
ISPRAS
Moscow
astrakhantsev@ispras.ru

© Denis Turdakov
ISPRAS
Moscow
turdakov@ispras.ru

© Natalia Vassilieva
HP Labs
Saint-Petersburg
vassilieva@hp.com

Abstract

This paper describes a novel approach to automate extraction of useful information from tables and to record the knowledge procured in a structured data repository. The approach is based on modeling a behavior of an expert, who collects tabular data and maps them to a predefined relational schema. Experimental results demonstrate that the proposed approach predicts expert decisions with high accuracy and thus significantly minimizes the time required of an expert for data aggregation.

1 Introduction

Tables are widely used in scientific, financial and other analytical documents to concisely communicate information to human readers. A table usually contains information about objects of the same type. These objects can be easily perceived by the reader through content and relations between different table elements (cells, rows, etc.), because he/she is experienced in table reading and is able to take in both semantic and structural information. Sometimes two tables with exactly the same structure are interpreted completely different just because of slight difference in the external context or because the content of some cells differs.

However, it is hard to automate table analysis and information is extracted mainly by hand by interested parties. A common scenario used by experts is manual cell by cell extraction of data from a table into a relational database, and then using OLAP or other techniques to generate reports and perform analytics over these data. Efforts required for manual extraction are considerable, while the time of experts is always costly.

There is no end-to-end solution for automatic information extraction from arbitrary tables. And as it appears to us, construction of a fully automatic instrument is hardly feasible. It might be possible to parse automatically the structure of any table, but semantic interpretation of tabular data requires the knowledge of a domain expert. Embley et al. [1] suggest The Periodic Table of the Elements as an example of the table requiring semantic knowledge for interpretation; the authors of this survey also state that it is easy to contrive other examples "that are challenging even from a human perspective".

We present a semi-automatic approach that tracks actions of a domain expert, when he/she begins to process a table (map cell data to a relational scheme), derives regularities/patterns of expert behavior, and applies them to the rest of the table in order to predict further mappings.

This paper is organized as follows: In Section 2 we give an overview of related works. Section 3 discusses the table format used in our work. Section 4 describes the general architecture of our prototype, while Section 5 describes it in details. Section 6 presents our experimental evaluation. We conclude in Section 7.

2 Related work

Silva et. al [11] survey about 50 works devoted to tables processing in details. Authors outline several table-related tasks and corresponding table representation models, which serve as input and output for these tasks. The span of tasks goes from location of a table in a document to semantic interpretation of the information contained in the table. There are more works focused on the basic low-level table related tasks than on the more knowledge based ones. A deep interpretation of the table is almost always requires context specific knowledge. Existing solutions for extracting information are very domain specific and designed for particular table types.

Zanibbi et. al [15] present the table recognition literature in terms of the interaction of table models, observations, transformations, and inferences. Most of described methods are fully automatic and consider the task of table processing in isolation from further usage of the extracted information.

Embley et al. [2] extract data from XML tables and map them to a given target database schema with 96/85 precision and 93/91 recall (depending on a domain — car advertisement and cell-phone correspondingly). However, their approach requires a hand-crafted ontology, which is costly and, more important, cannot always be in place due to high specificity of certain documents.

More recent work of Embley and Krishnamoorthy [3] transforms CSV or HTML tables

Proceedings of the 15th All-Russian Conference "Digital Libraries: Advanced Methods and Technologies, Digital Collections" — RCDL-2013, Yaroslavl, Russia, October 14-18 2013.

into a canonical representation in order to obtain a target representation, one of which is Relational table. A canonical table representation is based on Header Paths, a purely syntactic technique that relates headers and data cells. Further transformation into a target representation is performed by specially defined relational algebra. In contrast to the previous approach, this one does not use any semantic knowledge; also it demands a user to point out the top-left data cell in cases where proposed heuristics cannot define such a cell automatically. Precision of correct Header Paths construction is about 74%. In [8] the authors evolve this approach by using interactive tool VeriClick [9], 'a macro-enabled spreadsheet interface that provides ground-truthing, confirmation, correction, and verification functions for CSV tables'. However, this tool is used only to locate so-called 'critical cells': corner cells that allows to distinguish header and data regions in a table.

Vasudevan et. al [14] address a closely related task: to automate data extraction from financial reports presented in PDF documents with many tables. The proposed approach shows good results (95.7% precision and 78.4% recall), but it requires a quality review stage and, since it is based on domain knowledge heuristics, the range of its application is severely limited.

Gatterbauer et al. [4] extract information from web tables by using two-dimensional visual model provided by web browsers instead of tree-based (HTML) representation. Fumarola et al. [5] combine knowledge about the visual structure of the Web page and the HTML markup for web lists extraction. Their tool is applicable for web tables, too; moreover, the authors evaluate accuracy on the same dataset as Gatterbauer and report very good quality (more than 99% precision and recall on table records). But this dataset is domain-independent, and target format of table analysis is more general than ours; therefore, it cannot be directly compared with our method.

Looking at semi-automatic tools, Google Refine[1] should be noted. It is "a power tool for working with messy data, cleaning it up, transforming it from one format into another, extending it with web services, and linking it to databases like Freebase". However, Google Refine is mostly for data cleansing and is not capable of performing information extraction and interpretation. It also cannot track user decisions and anticipate them.

Similarly, Microsoft Excel[2] can perform such transformations using formulas and macros, but it demands user to define these formulas explicitly.

Praedea[3] is another semi-automatic tool that uses predefined text-mining models (chosen by user) for extracting required data from unstructured documents and mapping them to database or XML scheme. It is not capable to process a table in case there is no suitable predefined model for it.

Thus, except for the highly specialized programs like VeriClick and commercial tools like Google Refine or MS Excel, we are not aware of any research on interactive information extraction from tables; and the authors of VeriClick confirm this observation [5]. It also should be noted that most works try to convert a table into some more usable format, but not to extract needed parts of information from a table.

3 Table format

Like other basic notions, 'table' has a lot of different definitions.

Peterman et al. [10] suggest the following intuitive definition: "tables have a regular repetitive structure along one axis so that the data type is determined either by the horizontal or vertical indices." The definition given by Lopresti et al. [7] consists of similar items:

1. 2-D cell assembly for presenting information;
2. Regular, repetitive structure along at least one axis;
3. Datatype determined by either horizontal or vertical index.

Tijerino et al. [12] uses standard definition of a relational table.

In this work we consider a table to be a set of cells with some text content. In other words, it is the only property of a table that is used explicitly; other properties like repetitive structure or the same datatype in a column or a row are considered by our method implicitly.

Our prototype takes HTML tables as an input; therefore we use terminology from HTML in order to describe cell properties; for example, colspan as a relative width of a table cell. However, our method does not depend on any specific properties of HTML format.

We use classical spreadsheet addressing for cells. For example, A2 of Table 1 is an empty cell in the first column and the second row with rowspan equal 2.

Table 1: Example of a source table with spreadsheet coordinates

	A	**B**	**C**	**D**	**E**
1	Secret budget				
2		FY 2009		FY 2010	
3		Oper.	Capital	Oper.	Capital
4	HP	10	20	30	40
5	Oracle	50	60	10	20
6	Samsung	12	34	56	78

4 General architecture

Assume that a user processes a table step-by-step. During each *step* the user selects multiple cells from the table and maps them to a record in a relational database. Let us define *user feedback* as information about mappings from a single user step. See Table 2 for an example of the initial user feedback provided by him/her while processing Table 1.

1 http://code.google.com/p/google-refine

2 http://office.microsoft.com/en-us/excel/

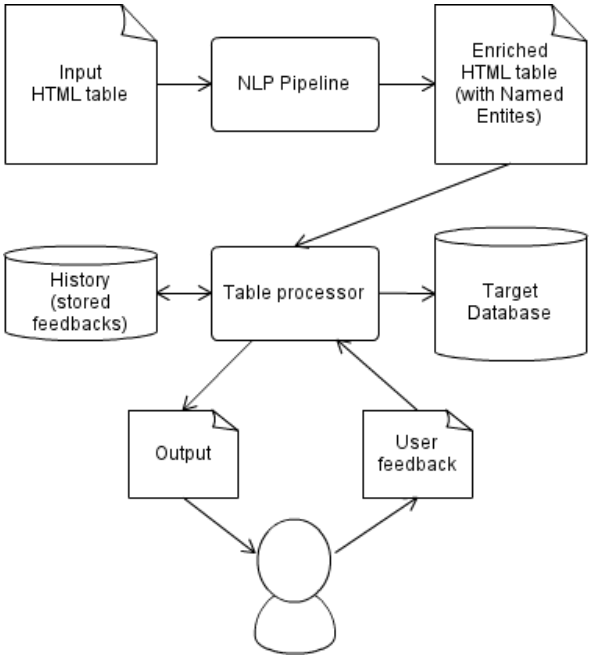
3 http://www.praedea.com

Table 2: User feedback example: two first columns represent data about the target relational scheme; two last ones represent table cell data

Relation name	Attribute name	Cell position	Value
Report	Company	A4	HP
Report	Operating	B4	10
Report	Financial Year	B2	2009

Figure 1 shows the general architecture of the prototype.

Figure 1: General architecture



The prototype takes as input an HTML table and enriches it by the predefined *Natural Language Processing pipeline*; currently we add only information about Named Entity types.

Enriched table is further processed by the main part of the prototype - *Table processor* on the diagram. It generates output, which has the same format as the user feedback, i.e. information about mappings that assumed to be obtained by the user. This output is reviewed by the user and he/she returns feedback. The prototype stores information from the user feedback into the *Target database* and uses this feedback in order to generate the next output. *History* means a local storage of the previous feedbacks, including ones from other tables that were already processed by the user. Information about previous actions is also used by the prototype for the output generation.

5 Shift approach

Our approach is based on two observations: (a) each table is a layout structure for storing similar objects; and (b) closely located tables (e.g. from the same document or Web-site) usually share similar structure and context. A user processes similar objects from the same table sequentially by repeating similar actions; and this repetition allows the system to learn and anticipate the forthcoming actions. The key idea of the approach is to consider the *shift* from one user step to the next one. More precisely, a shift consists of row and column offsets (possibly zero) for each cell represented in a pair of sequential user feedbacks. Since any table contains several objects of the same type and structure, we try to recognize it by tracking sequential user steps.

Table 3 shows an example of a shift from the first feedback (underlined words) to the second one (bolded words). Note that some cells remain fixed, e.g. *B2*.

Table 3: Shift example: from (HP, 10, 2009) to (Oracle, 50, 2009)

	A	B	C	D	E
1	Secret budget				
2		FY **2009**		FY 2010	
3		Oper.	Capital	Oper.	Capital
4	HP ↓	10 ↓	20	30	40
5	**Oracle**	**50**	60	10	20
6	Samsung	12	34	56	78

Shift approach is used in different ways depending on the number of the available user feedbacks. In section 5.1 we describe a regular phase of our algorithm: when at least two user feedbacks are available and we can explicitly compute shifts. Section 5.2 presents the phase when we have only one user feedback and have to guess a correct shift. Section 5.3 considers the phase when user starts to process a table and we have to predict his/her actions based on the previously processed tables.

5.1 Regular phase

After two user steps we can construct a shift by computing row and column offsets explicitly for every cell. For example, after two steps of processing Table 1 the following shift is constructed: the first two cells are shifted down by one cell and the third cell stays at the same position.

The constructed shift is further applied to cells of the last user step in order to get new cells and create a new row in a relational database with the same relation/attribute information and new values taken from new cells.

As it was shown in the example above, sometimes the value returned by the user is not just a text content of the cell, but its derivative: compare "*FY 2009"* in Table 1 with just "*2009"* in the feedback, Table 2. Solution of this problem is based on the observation that shifted cells share similar structure of their content. We store the way of producing the value from the cell in form of a regular expression. More precisely, we have prepared a set of patterns, which should be tested to match cell content against the value returned by the user within his/her feedback. Currently there are two types of patterns: (a) taking a substring of original cell content, and (b) replacing a substring of original cell content with another predefined string. Patterns of both types contain corresponding regular expressions to be applied to cell content.

Table 4: Example of cell value patterns

Pattern	Description	Cell value	Feedback value
.*::::0	Takes the value as is	HP	HP
(.*)\s(.*)::::1::::2	Splits by space and tries each part	Financial year	year
(\d+)::::1	Takes only digits from the content	146%	146
thousand::::000::::ReplaceAll	Replaces all words *thousand* by 3 zeros	45 thousand	45000

Patterns of the first type also contain matching group numbers to be taken into account; patterns of the second type contain a replacement string and a key word *ReplaceAll* in order to distinguish this type. See Table 4 for the examples of cell value patterns. Sequence of colons (::::) serves as a delimiter.

These patterns work as follows. Assume, have a cell with content *"FY 2009"* and the corresponding feedback value is *"2009"*. We try to apply the first pattern (.*) to cell content, which means that we simply take the whole content as is. The result does not match with the feedback value. Then we try the next pattern ((.*)\s(.*)::::1::::2). The result matches with the feedback value and we store this pattern for the corresponding cells of the shift.

If there are more than two steps produced by the user, then we construct candidate shifts for all paired combinations of user feedbacks, assess them, and choose the best shift in order to apply it to cells of the last user feedback. For example, if there are 3 feedbacks, we construct 3 candidate shifts: 1-to-2, 1-to-3, and 2-to-3; all 3 candidate shifts are assessed independently, and the best shift is then applied to the last, 3rd feedback.

Shift assessment is performed by computing linear combination of the following 4 features:

Average shift length: normalized sum of lengths of cells offsets. For the example shift, there are 2 offsets having length 2 and 1 offset having length 0, so the feature value is 2/3.

Motivation: people tend to process the table sequentially, taking closest cells if possible. In other words, shorter shifts are preferable by users.

We also tried different weights for different directions, e.g. bigger weight for the right offset than for the down one, because top-down processing seem to be more common, but experiments show that different weights do not introduce any positive effect.

Cells offsets consistency: normalized number of most common cells offsets. For the shift in the example, there are 2 types of offsets: 2 down-by-1 offsets and 1 remain-fixed offset, so the feature value is 2/3 again.

Motivation: a shift usually contains similar offsets of cells, e.g. all 4 cells are shifted down by 1 more often, then when 2 cells are shifted down by 1 and other 2 cells are shifted down by 2 or right by any number.

Average text similarity: normalized value of string similarity metrics computed over corresponding (shifted) cells contents.

Motivation: when we shift one cell to another, both of them must contain values of the same attribute, and different values of the same attributes are usually similar text strings.

To compute such similarity we tried several string metrics taken from SimMetrics[4], the best results have been obtained using Levenstein distance [6]. In addition, we modified string metric as follows:

1. All digits are considered to be equal characters, because, as it is written above, we want to capture strings of the same attribute, or data type, and difference between numeric strings tells nothing about difference between attributes. Note that we do not consider all numbers to be equal, because much difference in orders of magnitude can indirectly indicate different attributes.
2. If one of the strings is empty, we use pre-defined value (0.5): sometimes cell values are missed, for example, it can mean that the previous value or some default value should be taken instead. Unmodified metrics return zero similarity for such cases, but cell value absence does not necessarily indicate the difference in attributes.
3. If both strings are long (more than 3 words), we use pre-defined value (0.8): again, difference in long texts does not reflect difference in attributes.

Named entity type consistency: predefined value for 3 cases depending on named entity types of cells contents:

1. Named entity types are equal – value is 1;
2. Named entity types are unequal – value is 0;
3. Named entity types are both undefined – value is 0.5;

Motivation is the same as in the previous feature, but here we utilize information about named entity types in order to check attribute consistency. We use a conditional random field (CRF) model with a combination of different popular features applied in supervised named entity recognition [13]. There are 6 supported named entity types: Acronym, Date, Location, Numeric, Organization, Person.

Coefficients for the linear combination are chosen experimentally to maximize the accuracy: we test all possible values from 0 to 1 with step 0.2 so that their sum equals to 1. We found the best accuracy to be obtained with 0, 0.4, 0.2, 0.4 correspondingly; the further granulation does not change the result.

5.2 One User Feedback Phase

Given one user feedback we need a shift to apply it to this user feedback as it is done in Regular phase. There are two ways: take a most appropriate shift from the previously processed tables or construct a shift from

[4] SimMetrics is a Similarity Metric Library provided by UK Sheffield University
http://sourceforge.net/projects/simmetrics/

scratch. To find the most appropriate shift we check all stored shifts for applicability, and then assess similarity of all applicable shifts modified in accordance with the current table. To assess the modified shift we use the linear combination as in section 5.1, but coefficients are re-estimated (0.2, 0, 0.6, 0.2).

This way is very similar to No user feedback phase, see Section 5.3 for details. In short words, modification means that we replace contents of cells in the shift by the contents of the corresponding cells of the current table. For example, if there is a stored shift (from some previously processed table) with just one cell offset – C4-to-C5 with contents "USA"-to-"Russia" – then we modify the shift so that now contents of the cell offset is "20"-to-"60": we take contents of C4 and C5 cells of Table 1. Note that if the shift contains cell offset like B1-to-B2 with ordinary colspan and rowspan (all equal to 1), then such the shift is inapplicable for the Table 1, because B1 and B2 cells of this table have different colspans.

If there is no suitable shift, i.e. similarity of the best shift does not reach a predefined threshold, then we construct a new one. For this purpose, we iterate over combinations of all possible offsets of row and column for each cell. To limit combinatorial explosion we do not consider offsets above predefined thresholds: 5 for rows and 3 for columns. Each offset combination is actually a shift that can be assessed as it is described above; coefficients are left the same.

5.3 No User Feedback Phase

When no user feedback is available for the current table, we can use information about previously processed tables. We store all user feedbacks during table processing for the case if some of them have a structure (set of cell positions) appropriate for a new table. To choose the most appropriate user feedback we first check all of them for applicability, i.e. current table must have cells in all cell positions of the user feedback, and these cells must have the same characteristics — particularly, colspan and rowspan. Then all applicable user feedbacks are assessed by constructing and assessing special "fake" shifts from the stored feedback to the feedback obtained by applying the stored feedback structure to current table. Assume we have a feedback shown in Table 5 from the already processed Table 6 and we just begin to process Table 1.

Table 5: Example of stored feedback

Relation	Attribute	Cell	Value	Cell pattern
Report	Company	A4	Lenovo	.*::::0
Report	Capital	B4	105	.*::::0
Report	Market share	C4	33	(\d+)::::1

Then we take a structure of the stored feedback, that is actually a cell position and a cell pattern, and apply it to the considered Table 1, see Table 7.

Table 6: Source of stored feedback

	A	B	C
1	Public budget		
2	Company	Capital	Share of market
3	Dell	100	27%
4	**Lenovo**	**105**	**33%**

Table 7: Applied stored feedback

Relation	Attribute	Cell	Value	Cell pattern
Report	Company	A4	HP	.*::::0
Report	Capital	B4	10	.*::::0
Report	Share of market	C4	20	(\d+)::::1

After that we construct a shift from the stored feedback (Table 5) to the obtained feedback (Table 7) and assess it in the way described in section 3.1 with re-estimated coefficients (0.2, 0, 0.4, 0.4). Such assessment allows us to choose the most similar table among all already processed ones.

Note that the stored feedback from the 3rd row, but not 4th, is not applicable to Table 1, because A3 cells have different rowspans.

6 Evaluation

We compiled a set of 30 tables containing financial reports and a set of more than 150 corresponding user feedbacks[5]. We use the following test metrics: accuracy, precision and recall. Accuracy shows the fraction of outputs that fully match the user feedback: if at least one value in the tool output is wrong, the whole answer is considered to be wrong. Precision and recall characterize the number of correct cell mappings. Obviously, precision and recall can be much higher than accuracy, because many answers are partially correct.

Table 8 shows the results for the regular phase; the 2nd and the 3rd rows show efficiency of shift constructing module and shift choosing module respectively. We consider the shift constructor to work correctly if there is a right output (maybe not the chosen one); for the shift chooser we count only those outputs when there is a correct shift constructed and thereby the shift chooser had a chance to choose it.

Table 8: Regular phase results

Total accuracy	74%
Shift constructor accuracy	78%
Shift chooser accuracy	94%
Precision	84%
Recall	80%

Table 9 shows the results for the first 2 phases. Easy to see that these results depend on feedbacks from previously processed tables, because each user feedback is stored during the processing and affects the following outputs. However, the order of tables inside the

[5] Dataset's URL : http://modis.ispras.ru/datasets/td.zip

document may be valuable; therefore, we shuffle order of blocks containing tables from the same document.

It is worth mentioning that the results of the first two phases also depend on stored shifts (feedbacks, for the first phase) and at the begging of the work, without any stored shift, we face the problem of cold start. That is why we add tests with prepared set of 5 simple stored feedbacks and shifts from other tables.

In addition, we also run tests when all feedbacks and shifts from the same 30 test tables are stored and used for the testing. Of course, it is not an absolutely fair test, because there are stored feedbacks and shifts with strictly the same text content, but these tests may help to understand if the mistakes are caused by the problem of missing stored shifts or the incorrect choice of the stored shift to be applied.

Table 9: Results for No user feedback and One user feedback phases

Phase	Number	Accuracy	Precision	Recall
No user feedback	0	4%	11%	7%
	5	4%	8%	6%
	all	16%	28%	25%
One user feedback	0	43%	91%	90%
	5	48%	87%	86%
	all	86%	87%	86%

Table 10 shows results for each module for one user feedback phase. The similarity estimator chooses the most similar shift among all stored ones. The similarity threshold is estimated perfectly: if the similarity estimator chooses an appropriate shift among the stored ones, then we always choose it and never try to construct our one instead. The shift constructor works not bad, it means that our algorithm constructs most of possible shifts, but the shift chooser makes a lot of mistakes.

Table 10: Accuracy of modules for one user feedback phase

Evaluated module	0 stored shifts	5 stored shifts	All stored shifts
Similarity	100%	100%	100%
Similarity	100%	100%	100%
Shift chooser	25%	20%	0%
Shift	80%	77%	33%

Some mistakes in the regular phase could be explained by the following: sometimes a user goes from the top to the bottom of the left part of the table (e.g. takes all values from the first and the second columns) and then he/she repeats the similar actions for the right part (takes all values from the first and the third columns). Our tool cannot predict a correct shift at the moment when the user switches to the right part, that is why results for shift constructor are low.

7 Conclusions and Future Work

In this paper, we focused on the task of semi-automatic data extraction from tables and mapping them to relational scheme; we introduce novel approach that tracks user decisions to predict forthcoming ones; we evaluated it on our test data.

Shift approach provides general scheme for semi-automatic table processing, but many problems are out of scope of this research. One of them is extraction of value from table cell. In most cases it is sufficient to take text substrings (see cell B2 in Table 1), but sometimes certain table cell is actually a set of different attribute values that should be processed by more complex methods. For instance, a price list of a hardware store often contains cells like the following:

HP "Pavilion dm4-2102er" QJ453EA (Core i5 2430M-2.40GHz, 6144MB, HD6470M, WebCam)

Another direction of further research is related to target scheme: currently we copy information about relation and attribute for shifted cells, but methods that are more sophisticated can consider semantics of both table content and relational scheme.

References

[1] D. W. Embley, M. Hurst, D. Lopresti, G. Nagy. Table-processing paradigms: a research survey. International Journal of Document Analysis and Recognition (IJDAR), 8(2-3), p. 66-86, 2006.

[2] D. W. Embley, C. Tao, S. W. Liddle. Automating the Extraction of Data from HTML Tables with Unknown Structure. Knowledge Engineering, 54 (1), p. 3-28, 2005.

[3] D. W. Embley, M. Krishnamoorthy. Factoring Web Tables. Proceedings of 24th International Conference on Industrial Engineering and Other Applications of Applied Intelligent Systems, p. 253-263, 2011.

[4] F. Fumarola, T. Weninger, R. Barber, D. Malerba, and J. Han. HyLiEn: a hybrid approach to general list extraction on the web. Proceedings of the 20th international conference companion on World wide web, pp. 35–36, 2011.

[5] W. Gatterbauer, P. Bohunsky, M. Herzog, B. Krüpl, and B. Pollak. Towards domain-independent information extraction from web tables. Proceedings of the 16th international conference on World Wide Web, pp. 71–80, 2007.

[6] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. Soviet Physics Doklady, 10: p. 707–10, 1966.

[7] D. Lopresti, G. Nagy. A tabular survey of automated table processing. Graphics Recognition Recent Advances, p. 93-120, 2000.

[8] G. Nagy, S. Seth, D. Jin, D. W. Embley, S. Machado, and M. Krishnamoorthy. Data extraction from web tables: The devil is in the details. Document Analysis and Recognition (ICDAR), pp. 242–246, 2011.

[9] G. Nagy and M. Tamhankar. VeriClick: an efficient tool for table format verification. IS&T/SPIE Electronic Imaging, p. 82970M–82970M, 2012.

[10] C. Peterman, C.H. Chang, H. Alam. A system for table under-standing. Proceedings of the Symposium on Document Im-age Understanding Technology (SDIUT'97), p. 55–62, 1997.

[11] A. C. Silva, A. M. Jorge, L. Torg. Design of an end-to-end method to extract information from tables. International Journal on Document Analysis and Recognition (8), No. 2-3, p. 144-171, 2006.

[12] Y. A. Tijerino, D. W. Embley, D. W. Lonsdale, Y. Ding, G. Nagy. Towards ontology generation from tables. World Wide Web 8, no. 3, 261-285, 2005.

[13] M. Tkachenko, A. Simanovsky. Named entity recognition: Exploring features. Proceedings of KONVENS 2012, p. 118-127, 2012.

[14] B. G. Vasudevan, A.G. Parvathy, A. Kumar, R. Balakrishnan. Automated Knowledge-based Information Extraction from Financial Reports. Knowledge Engineering and Management, 7(5), p. 61-68, 2009.

[15] R. Zanibbi, D. Blostein, J. R. Cordy. A survey of table recognition: Models, observations, transformations, and inferences. International Journal of Document Analysis and Recognition, 7(1), Springer, Heidelberg, p. 1–16, 2004.