Populating Ontologies with Data from OCRed Lists

Thomas L. Packer
Brigham Young University
Provo, Utah, USA
Email: tpacker@byu.net

David W. Embley
Brigham Young University
Provo, Utah, USA
Email: embley@cs.byu.edu

***Abstract*—A flexible, accurate, and efficient method of automatically extracting facts from lists in OCRed documents and inserting them into an ontology would help make those facts machine searchable, queryable, and linkable and expose their rich ontological interrelationships. To work well, such a process must be adaptable to variations in list format, tolerant of OCR errors, and careful in its selection of human guidance. We propose a wrapper-induction solution for information extraction that is specialized for lists in OCRed documents. In this approach, we induce a regular-expression grammar that can infer list structure and field labels in sequences of words in text. We decrease the cost and improve the accuracy of this induction process using semi-supervised machine learning and active learning, allowing induction of a wrapper from a single hand-labeled instance per field per list. To further reduce cost, we use the wrappers learned from the semi-supervised process to bootstrap an automatic (weakly supervised) wrapper induction process for additional lists in the same domain. In both induction scenarios, we automatically map labeled text to a rich variety of ontologically structured facts. We evaluate our implementation in terms of annotation cost and extraction quality for lists in historical documents.**

I. Introduction

Family history books and other machine-printed documents present much of their valuable content in data-rich lists. The 50,000+ family history books held by FamilySearch.org are full of lists containing hundreds of millions of fact assertions about people, places, and events. Figure 1 shows examples of lists found on Page 154 of *The Ely Ancestry* [2]. These lists make many assertions about family relationships (Samuel Holden Parsons was a child of Deborah Mather and Ezra Lee) and dates, and of life events (Samuel Holden Parsons was born in 1772 and died in 1870). Our goal is to develop a means to extract the diverse kinds of facts from lists in OCRed documents that is robust to OCR errors and relies on as little human effort as possible.

To be most useful to downstream search, query, and data-linking applications, the knowledge extracted from text must be expressive and well structured. Ontologies are a machine-readable and mathematically specified conceptualization of a collection of facts. They are expressive enough to provide a framework for storing more of the kinds of assertions found in lists than the typical output of named entity recognition and most other information extraction work. If we could populate user-specified ontologies with predicates representing the facts in OCRed lists, this richer, more expressive, and versatile information could better contribute to a number of applications in historical research, database querying, record linkage, automatic construction of family trees, and question answering.

1555. Elias Mather, b. 1750, d. 1788, son of Deborah Ely and Richard Mather; m. 1771, Lucinda Lee, who was b. 1752, dau. of Abner Lee and Elizabeth Lee. Their children:—

1. Andrew, b. 1772.
2. Clarissa, b. 1774.
3. Elias, b. 1776.
4. William Lee, b. 1779, d. 1802.
5. Sylvester, b. 1782.
6. Nathaniel Griswold, b. 1784, d. 1785.
7. Charles, b. 1787.

1556. Deborah Mather, b. 1752, d. 1826, dau. of Deborah Ely and Richard Mather; m. 1771, Ezra Lee, who was b. 1749 and d. 1821, son of Abner Lee and Elizabeth Lee. Their children:—

1. Samuel Holden Parsons, b. 1772, d. 1870, m. Elizabeth Sullivan.
2. Elizabeth, b. 1774, d. 1851, m. 1801 Edward Hill.
3. Lucia, b. 1777, d. 1778.
4. Lucia Mather, b. 1779, d. 1870, m. John Marvin.
5. Polly, b. 1782.
6. Phebe, b. 1783, d. 1805.
7. William Richard Henry, b. 1787, d. 1796.
8. Margaret Stoutenburgh, b. 1794.

Fig. 1. Lists in *The Ely Ancestry*, Page 154

In this paper we propose ListReader, a robust, general, and efficient solution to the challenge of extracting diverse types of facts from lists in OCRed documents. We have implemented and tested an early prototype of ListReader that populates a user-defined ontology with the assertions found and labeled automatically in OCRed lists. A ListReader user constructs an ontology for a list by building a data-entry form in a custom web interface and fills in the form with the information from the first record of a list. Taking this minimal amount of information, ListReader induces a regular-expression wrapper and automatically generalizes it enough to extract asserted information from the remaining records of the list. Only when ListReader encounters a new field in a later record must it ask the user to update the form to accommodate the new field and insert the field value to provide additional training data. This is the minimum amount of training data conceivable as the user begins to train ListReader to recognize information in a new domain and document type. We call this process semi-supervised wrapper induction. After ListReader has begun inducing grammars and extracting information from a document, it can switch into a weakly supervised or transfer learning mode in which it uses its store of knowledge to effectively label its own training data for other lists, potentially removing the human user from the process.

Researchers have studied information extraction [17], and wrapper induction as an approach to information extraction [7], [13], [16], for more than a decade. Similar to the wrapper-induction work of Dalvi, et al. [7] and Kushmeric [13], we attempt to improve efficiency in terms of human effort

by inducing wrappers from training data that is generated automatically, given little if any domain knowledge. However, unlike our work, theirs relies on the layout of HTML text that is relatively regular and consistent, not considering list records containing OCR errors and frequently missing fields, and lacking consistent field landmarks. Further, their work relies on pre-assembled dictionaries and regular-expression recognizers, which we avoid.

Other information extraction papers target lists [9], [10], [14], but very few target OCRed lists. Those that target non-OCRed lists target HTML lists and generally rely on consistent landmarks (e.g. HTML tags) that are not available in OCR text. Furthermore, and perhaps more importantly relative to our work, they do not target the diverse semantic distinctions in the rich ontological structures that we do. Most information-extraction work targeting OCRed lists is specific to certain kind of lists. Belaïd [3] [4] and Besagni, et al. [5] [6] extract records and fields from lists of citations, but rely primarily on hand-crafted knowledge that is specific to bibliographies. A paper by Adelberg [1] and one by Heidorn and Wei [12] target lists in OCRed documents in a general sense. They, however, use supervised wrapper induction that we believe will not scale as well as our semi-supervised or weakly supervised approaches when encountering the "long tail" of list formats. Also, the extracted information is limited in ontological expressiveness.

In this paper, we make the following contributions. (1) We establish a formal correspondence among list wrappers, ontologies, data-entry forms, and in-line annotated text. This correspondence provides the data flow for a processes in which a user can annotate OCRed text as training data for wrapper induction. It also enables even simple induced wrappers that produce in-line or sequentially labeled text to extract rich facts from lists and insert them into an expressive ontological structure (Section II-A). (2) We demonstrate that it is possible to perform wrapper induction for a list using only one human-provided label per field (Section II-B). (3) We show one way that automatic labeling can replace a human labeler in providing input to wrapper induction by using wrappers previously induced from other lists (Section II-C). (4) We evaluate extraction accuracy and show that ListReader outperforms a general, state-of-the-art information extraction system with high statistical significance (Section III). (5) We conclude that we can benefit from a new line of research, state limitations of our current approach, and identify opportunities for future research into cheaply inducing accurate wrappers for general OCRed lists (Section IV).

II. List Wrapper Induction

A. ListReader Overview

ListReader populates an ontology from lists in OCRed text as follows:

First, a user selects an OCRed image (an OCRed PDF page in our implementation) that contains a list (e.g. Figure 1). Initially, when ListReader has no information in its knowledge repository that would allow it to find and process lists on its own, a user spots a list and, with ListReader's form interface, constructs a form for the data fields in the first record of the spotted list and fills in the form with text from that first record. For example, supposing the spotted list is the second child list in Figure 1, the user would construct the form in Figure 2 and fill it in—copying and pasting from the PDF page.

Second, from the empty form, ListReader creates the schema of an ontology (e.g. Figure 3 for our example). It can also populate the ontology with the information in the filled-in form, but its main objective is to induce a wrapper for the list and use the wrapper to populate the ontology with all the fact predicates stated in the list.

Third, to induce a wrapper, ListReader uses the information obtained from the filled-in form to label the fields within the OCRed text as training data. Figure 4 shows the labeled text for our example, which ListReader automatically labels and processes as follows.

1) Because the form is filled in using a custom user interface, ListReader knows the character offsets of the text string for each field.
2) ListReader labels text strings within the OCRed document with path expressions. Each path refers to a path in the ontology hyper-graph from the root node to a leaf node. The leaf node holds the extracted text string as a named entity and corresponds to the field in the filled-in form. The root node is the primary node in the ontology, whose name is same as the form title, that represents the main concept of each record of the list (*Person*, in Figure 2). Thus, for example, the full label for the year of Samuel's birth is <*Person.BirthDate.Year*>, which in Figure 4 appears in its abbreviated form as <*BirthDate.Year*>—with prefix path segments omitted when there is no ambiguity.
3) The in-line labeled text provides enough information to map stated facts to and from a filled-in form and an ontology populated with fact predicates.
4) The in-line labeled text also provides enough information for ListReader to induce an information extraction wrapper for the whole list, effectively reducing the ontology populating problem to a machine-learned sequential labeling problem as we explain in Section II-B.

Fourth, the induced wrapper labels the remaining records in the list with labels like those provided in its training data.

Finally, ListReader saves the induced wrapper and ontology in its knowledge repository and uses it to find and process similar lists in other OCRed document images. In this case, the user needs neither to create a form to generate the ontology nor to fill in the form to label any of the fields of any of the records. We explain how this works in Section II-C.

The formal correspondence among form, ontology, and text labels provides for, on the one hand, ease of use for ontology creation and annotation of text, and, on the other hand, inducing a wrapper from labeled training data such that when executed it can properly map identified instances to the ontology structure. The metaphor of form fill-in for obtaining information is familiar to most users as is form creation from the set of primitives we provide: single-entry blanks (e.g. *Year* in Figure 2), multiple-entry blanks (e.g. *Name*), and check boxes and radio buttons for role designators (e.g. *Child*). Form-field nesting is fully recursive so that any form structure can be nested within any other structure to any depth.

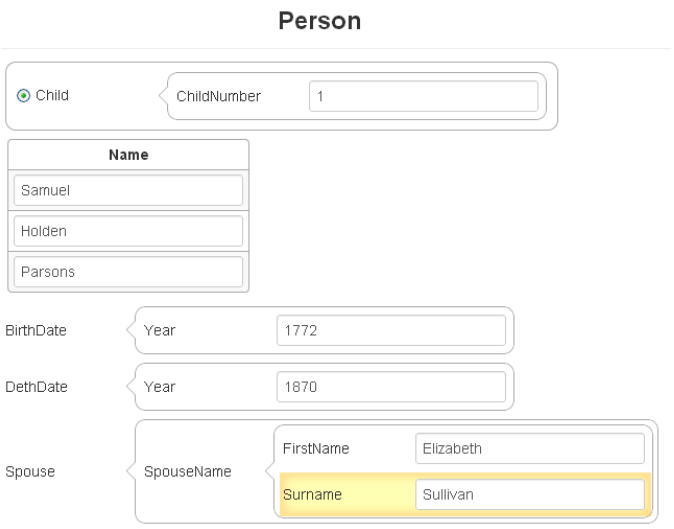

Fig. 2. Filled in Form for Samuel Holden Parsons Record

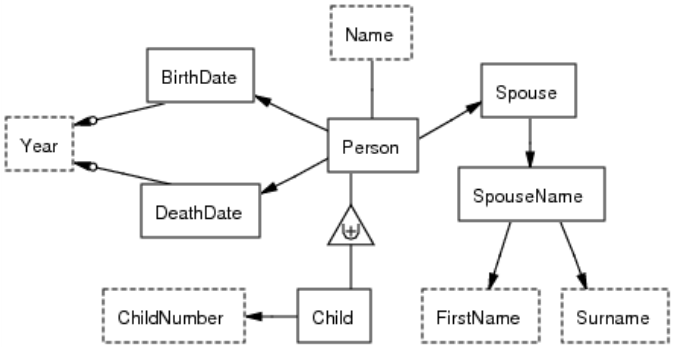

Fig. 3. List Ontology for Samuel Holden Parsons List

These form primitives, along with nesting, provide for a rich set of ontological structures. Each named form primitive corresponds to an object set, and each nesting corresponds either to to a relationship set or to a role specialization. Object sets are unary predicates and relationship sets are n-ary predicates (in practice most relationship sets are binary, $n = 2$, but $n > 2$ is also possible). Instantiated predicates are fact assertions. In connection with our claim of ontological richness in the kinds of facts ListReader can extract, we mention five points of expressiveness: (1) textual vs. abstract entities (e.g. *Name*("Elias") vs. *Person*(p_1)), (2) n-ary relationships among two or more entities instead of strictly unary and binary relationships (e.g. Husband-married-Wife-in-Year(p_1, p_2, "1771")), (3) ontology hypergraph with arbitrary path lengths from the root instead of just one as in named entity recognition or data frame filling (e.g. <Person.Spouse.SpouseName.Surname>, (4) functional and optional constraints on relationship sets (e.g. A person has one birth event vs. zero or more marriage events), (5) generalization-specialization hierarchies, including, in particular, role designations (e.g. *Child isa Person*).

B. Semi-supervised Wrapper Induction

In the semi-supervised wrapper induction setting, ListReader begins learning from nothing more than the text

```
<ChildNumber>1</ChildNumber>. <Name>Samuel</Name>
 <Name>Holden</Name> <Name>Parsons</Name>
, b. <BirthDate.Year>1772</BirthDate.Year>
, d. <DeathDate.Year>1870</DeathDate.Year>
, m. <FirstName>Elizabeth</FirstName>
 <Surname>Sullivan</Surname>.
```

Fig. 4. Labeled Samuel Holden Parsons Record

Label	Initial Regex	Final Regex RecordType1	Final Regex RecordType2
RecordDelimiter	(\n)	(\n)	(\n)
ChildNumber	(\d)	(\d)	(\d)
FieldDelimiter	(\.\s)	(\.\s)	(\.\s)
Name	(\w{6,6})	(\w{5,9})	(\w{5,9})
FieldDelimiter			(\s)
Name			(\w{3,8})
FieldDelimiter	(,\sb\.\s)	(,\sb\.\s)	(,\s[bh]\.\s)
BirthDate.Year	(\d{4,4})	(\d{4,4})	([i0-9]{4,4})
FieldDelimiter			([\.,]\sd\.\s)
DeathDate.Year			(\d{4,4})
FieldDelimiter	(\.)	(\.)	(\.)
RecordDelimiter	(\n)	(\n)	(\n)

Fig. 5. Regex Induction for First Child List in Fig. 1

of an OCRed page image with the fields of the first record of a list labeled. ListReader initializes a new wrapper to model the text and labels of the first record using a sequence of capture groups corresponding to the sequence of fields and delimiters. Consider, for example, the following labeling of the first record of the first child list in Figure 1:

```
<ChildNumber>1</ChildNumber>. <Name>Andrew</Name>,
b. <BirthDate.Year>1772</BirthDate.Year>
```

From this labeling, ListReader generates the initial regular expression (regex) in Figure 5. The field delimiters are exactly those sequences of characters that appear between the labeled fields, and the fields are a first level generalization of the field content itself—a sequence of word characters of the observed length or a sequence of digits of the observed length.

ListReader generalizes the initial regex in four steps to produce a set of regexes, one for each record type. In the first step, ListReader applies a predefined set of regex expansion operators to enumerate a space of candidate regexes. These operations include: (1) replace a word appearing in a labeled field with a more general character class to account for both intentional content variations and OCR errors (e.g. replace "Samuel" with "\w{6,6}"), (2) replace each character in a field delimiter with a predefined set of common OCR error substitutions (e.g. replace "[\.]" with "[\.,]"), (3) generalize word length (e.g. replace "\w{6,6}" with "\w{3,10}"), (4) increase (or decrease) the length of a field by adding (or removing) space-delimited words, (e.g. "replace (\w{3,10})" with "(\w{3,10}\s\w{3,10})"), (5) delete a field and its neighboring delimiters, and (6) insert a new capture group containing between one and four words (e.g. "(\S{1,10}\s\S{1,10})"). ListReader assigns a field label of "Unknown" to inserted capture groups. Notice that we do not generalize field delimiters as much as field content since delimiters should remain constant, while field values should differ. ListReader applies all these operators in combination, each a bounded number of times.

In the second step, ListReader scores and ranks the candidate regexes. ListReader assigns a quality score to each regex

which is the product of *similarity* and *match-frequency* (all values are between 0.0 and 1.0). *Similarity* is 1 minus the edit-distance from the initial regex. (Values less than zero are set to 0.0.) We compute edit-distance by summing an empirically determined distance or cost value assigned to each operator (e.g. 0.0 for an OCR error character expansion and 0.1 for adding a new capture group). *Match frequency* is the number of candidate records matched in unlabeled text divided by an empirically determined maximum number of records that a single regex is expected to match. (Values above 1.0 are set to 1.0.)

In the third step, ListReader determines whether any active learning is necessary and interacts with the user when it is. ListReader executes candidates against the unlabeled text in the order of their quality scores and removes segments of text that match. If ListReader encounters a regex with an "Unknown" label while removing records, it alerts the user by highlighting the section of text matched by the "Unknown" capture group. The user may then modify the form, which provides a label name for the field and updates the ontology, and then copy the part of the highlighted section that constitutes the field value into the new form field. ListReader then regenerates that section of the matching regex. For example, the regex in the last column of Figure 5 begins as the initial regex with two "Unknown" capture groups that match " Lee" and ", d. 1802" in the fourth record in the first list of Figure 1. The user would then add fields to the form for both, inserting "Lee" into an additional *Name* field and "1802" into a new *DeathDate.Year* field. ListReader then adjusts the regular expression by replacing the "Unknown" capture groups with, in this instance, (\s)(\w{3,3}) and (,\sd\.\s)(\d{4,4}) respectively.

In the last step, ListReader makes the final set of regexes no more general than necessary to match the text. It then stores the induced wrapper in its knowledge repository for use in weakly supervised wrapper induction (Section II-C). Reducing wrappers to just what's necessary helps prevent overgeneralization. The field-value length for the first *Name* field in Figure 5, for example, is initially (\w{6,6}), which ListReader generalizes to (\w{3,10}) before finally replacing it with (\w{5,9}) to match the actual name lengths that appear. For OCR errors, instead of the full set of possible errors from a confusion matrix for "b.", for example, ListReader cuts the set back to just "[bh]" for "b" and [\.,] for "." if these are the only actual OCR errors encountered.

After inducing a wrapper and labeling the records of a list, ListReader uses the labels to instantiate the generated ontology for the list. For the labeled field values in a record, ListReader creates objects and relationships corresponding to each label's path and properly links the field values within the record according to the structure of the ontology.

C. Weakly-supervised Wrapper Induction

In the weakly-supervised setting, ListReader begins wrapper induction with a completely unlabeled page and a non-empty repository of induced wrappers and corresponding populated ontologies. To produce a labeled record needed to start wrapper induction, ListReader applies the stored wrappers, and also further-generalized versions of those wrappers, to the page. ListReader creates its own training data by selecting the labels produced by the best regex, where "best" means the highest product of the regex quality score and the number of labeled fields within the regex. ListReader then executes the semi-supervised wrapper induction process (Section II-B), looking both above and below the starting record.

To illustrate, consider applying the wrapper in Figure 5 (induced for the first list in Figure 1) to the second list in Figure 1. The regex for Record Type 1 in Figure 5 immediately matches Record 5 in the second list, and the regex for Record Type 2 with the deletion of the second name matches Records 3 and 6 and with the deletion of the death date and a generalization of the length of the second name matches Record 8. ListReader can start from any one of these records and, with active learning, acquire the additional fields needed to induce a wrapper for the second list. On the other hand, suppose a wrapper for the second list in Figure 1 were generated first. This is a more interesting situation because ListReader can then induce a wrapper for the first list without user intervention. A regex, with a name length generalization and a deletion of the marriage information of the induced regex for Record 4 in the second list, matches both Record 4 and 6 in the first list. Then, ListReader can induce, on its own, the regex for the remaining records in the first list by deletion of the second name and the death date. Generation of the ontology for the list is also automatic. For this example, it is the ontology in Figure 3 with all the spouse information deleted.

III. Experimental Evaluation

A main objective of developing ListReader is to find a way to reduce cost (human labeling) and increase accuracy (F-measure) for inducing wrappers for lists by taking advantage of list structure. Since we can view ListReader as a machine-learned sequential labeler, we empirically compare it to a highly regarded Conditional Random Field (CRF) statistical sequence labeler [15]. To make our labeling task learnable by the CRF and to ensure a fair test, we tuned its hyper-parameters and selected an appropriate set of word features. We also tuned ListReader's set of regex operators and scoring function, tuning both the CRF and ListReader on the same separate development data. As test data, we randomly selected and isolated the text of 30 child lists[1] from throughout *The Ely Ancestry* [2] containing a total of 137 records. We compute F-measures for field labels over all word tokens not used as hand-labeled training data. All reported differences between CRF and ListReader F-measures are statistically significant at the $p < 0.01$ level using McNemar's test [8].

To test ListReader's semi-supervised wrapper induction, we hand-labeled the first record of each list and ran ListReader separately on it. We compare the results of ListReader to the CRF, also run separately on each list with varying amounts of training data. Table I shows the results. Hand-labeling just the first record has a lower cost for the CRF compared to ListReader (4.4 v. 5.9), but the F-measure is lower. Not until trained with the "three best" records does the F-measure of the CRF approach that of ListReader. Even then it is still significantly less and at almost double the number of labels plus the effort to select the "three best"—a combination of a

[1] Some list text in the *The Ely Ancestry* varies more than our current implementation of ListReader is designed to handle (see limitations in Section IV). We discarded these lists and kept selecting until we obtained 30.

TABLE I. SEMI-SUPERVISED RESULTS

	Accuracy			Cost
	P.	R.	F.	# Labels / List
ListReader	98%	84%	90%	5.9
CRF 1^{st} Rec.	86%	71%	78%	4.4
CRF Best Rec.	86%	70%	77%	4.7
CRF Best 2	87%	82%	84%	8.4
CRF Best 3	87%	86%	87%	11.0

TABLE II. WEAKLY-SUPERVISED RESULTS

	Accuracy			Cost
	P.	R.	F.	# Labels / List
ListReader	94%	59%	72%	0.53
CRF	72%	68%	69%	0.66

longest (1^{st} Best), a least typical (2^{nd} Best), and a most typical (3^{rd} Best) record.

To evaluate weakly supervised learning, we ran ListReader on one of the lists in semi-supervised mode and then executed ListReader in weakly-supervised mode on each one of the remaining 29 lists. We were thus able to see how well ListReader could use a wrapper generated for one list to identify and label a starting record in another list and induce a wrapper for it with no more human labeling than for new unique fields not found in the first list. For the CRF, we hand-labeled all records in one list to train it and then executed it on each of the remaining 29 lists. For both ListReader and the CRF, we repeated the procedure 30 times, using each list in our test set as a starting list, to compute the averages in Table II. ListReader achieves a higher F-measure at a lower cost than the CRF, although the low F-measure indicates there is still room for improvement.

IV. CONCLUSIONS AND FUTURE WORK

These encouraging early results suggest that ListReader is a viable new line of research for solving the general problem of populating ontologies with data from lists in OCRed documents. ListReader outperforms a CRF ($p < 0.01$) in all our tests showing that we can better leverage the characteristics of list structure with a more tailored machine learning approach. ListReader has the potential to reduce human effort, not only limiting user involvement to labeling each distinct field of a list only once, but for an entire collection of lists, like the child lists in *The Ely Ancestry*. Additionally, ListReader uniquely obtains rich ontological assertions that are more useful than simply labeling words with named entity categories.

To continue this line of research, we will address a few current limitations. When executed against lists containing greater variation between records than two field insertions/deletions, several-word (>4) insertion fields, or parenthetical comments, the current approach of exhaustively enumerating candidate regexes fails to scale in both time and space and becomes sensitive to parameter settings. We plan to investigate three alternative designs: (1) record-incremental search in which ListReader adjusts the wrapper one record at a time, (2) A* search [11] using a specially designed admissible heuristic allowing only the most promising intermediate regexes to be expanded, and (3) a Hidden Markov Model whose structure is learned methodically from one labeled field per list.

In addition to investigating alternative wrapper induction algorithms, we intend to work with more complex lists: (1) lists split by intervening text or page breaks (e.g. the lists in *The Ely Ancestry* that split across page boundaries), (2) lists nested within other lists (e.g. the nested child lists in the larger family list in Figure 1), (3) lists with fields factored out of each record, (e.g. the surname of the children in a family factored out of the child lists in Figure 1), (4) lists whose records describe entities from distinct categories (e.g. child lists containing records with distinct structures for sons and daughters), and (5) lists that can be modeled by joining fragments of previously learned wrappers and ontologies (e.g. parish christening records learned from joining parts of family lists with parts of church administration lists).

REFERENCES

[1] B. Adelberg. NoDoSE — a tool for semi-automatically extracting structured and semistructured data from text documents. *ACM SIGMOD Record*, 27:283–294, 1998.

[2] M. S. Beach, W. Ely, and G. B. Vanderpoel. *The Ely Ancestry*. The Calumet Press, New York, New York, USA, 1902.

[3] A. Belaïd. Retrospective document conversion: application to the library domain. *International Journal on Document Analysis and Recognition*, 1:125–146, 1998.

[4] A. Belaïd. Recognition of table of contents for electronic library consulting. *International Journal on Document Analysis and Recognition*, 4:35–45, 2001.

[5] D. Besagni and A. Belaïd. Citation recognition for scientific publications in digital libraries. In *Proceedings of the First International Workshop on Document Image Analysis for Libraries*, pages 244–252, Palo Alto, California, USA, 2004.

[6] D. Besagni, A. Belaïd, and N. Benet. A segmentation method for bibliographic references by contextual tagging of fields. In *Proceedings of the Seventh International Conference on Document Analysis and Recognition*, pages 384–388, Edinburgh, Scotland, 2003.

[7] N. Dalvi, R. Kumar, and M. Soliman. Automatic wrappers for large scale web extraction. *Proceedings of the VLDB Endowment*, 4:219–230, 2010.

[8] T. G. Dietterich. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, 10(7):1895–1923, Oct. 1998.

[9] H. Elmeleegy, J. Madhavan, and A. Halevy. Harvesting relational tables from lists on the web. *Proceedings of the VLDB Endowment*, 2:1078–1089, 2009.

[10] R. Gupta and S. Sarawagi. Answering table augmentation queries from unstructured lists on the web. *Proceedings of the VLDB Endowment*, 2:289–300, 2009.

[11] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[12] P. B. Heidorn and Q. Wei. Automatic metadata extraction from museum specimen labels. In *Proceedings of the 2008 International Conference on Dublin Core and Metadata Applications*, pages 57–68, Berlin, Germany, 2008.

[13] N. Kushmerick. *Wrapper induction for information extraction*. PhD thesis, University of Washington, Seattle, Washington, USA, 1997.

[14] K. Lerman, C. Knoblock, and S. Minton. Automatic data extraction from lists and tables in web sources. In *IJCAI-2001 Workshop on Adaptive Text Extraction and Mining*, volume 98, 2001.

[15] A. K. McCallum. MALLET: a machine learning for language toolkit. http://mallet.cs.umass.edu/, 2002.

[16] I. Muslea, S. Minton, and C. Knoblock. Stalker: Learning extraction rules for semistructured, web-based information sources. In *Proceedings of AAAI-98 Workshop on AI and Information Integration*, page 74–81, 1998.

[17] S. Sarawagi. Information extraction. *Foundations and Trends in Databases*, 1:261–377, 2008.