Improved bibliographic reference parsing based on repeated patterns

Guido Sautter & Klemens Böhm

International Journal on Digital Libraries

ISSN 1432-5012

Int J Digit Libr
DOI 10.1007/s00799-014-0110-6

Springer

Int J Digit Libr
DOI 10.1007/s00799-014-0110-6

Improved bibliographic reference parsing based on repeated patterns

Guido Sautter · Klemens Böhm

Received: 31 October 2012 / Revised: 30 January 2014 / Accepted: 6 February 2014

Abstract Parsing details like author names and titles out of bibliographic references of scientific publications is an important issue that has received considerable attention recently. However, most existing techniques are tailored to the highly standardized reference styles used in the last two to three decades. They do not perform well with the wide variety of reference styles used in older, historic publications. Thus, they are of limited use when creating comprehensive bibliographies covering both historic and contemporary scientific publications. This paper presents a generic approach to bibliographic reference parsing, named RefParse, which is independent of any specific reference style. Its core feature is an inference mechanism that exploits the regularities inherent in any list of references to deduce its format. In addition, our approach learns names of authors, journals, and publishers to increase the accuracy in scenarios where human users double check parsing results to increase data quality. Our evaluation shows that our approach performs comparably to existing ones with contemporary reference lists and also works well with older ones.

Keywords Parsing · Bibliography data · Algorithms

1 Introduction

Bibliographies covering both historic and contemporary publications in comprehensive ways are more and more important in many scientific domains. In biodiversity, for instance, some currently ongoing projects [8,25] aim at creating such bibliographies. While publication indexing services like Citeseer[1] [6] and Google Scholar[2] are very helpful with contemporary documents, they are less useful with historical ones, which might not even be available in digital form. However, many historic publications have been cited many times, so their metadata are available from the bibliographies of other publications. Thus, there is a good chance to find a digital version of at least some of the metadata.

Extracting bibliographic references in a more fine-grained form than plain strings requires parsing. In particular, this includes identifying individual attributes of the referenced work, e.g., the paper title and the names of the authors. Only this level of detail enables advanced processing of the data, for instance detailed search, or identification and reconciliation of different reference strings that point to the same work [12,14]. Examples of other, yet higher-level applications are the disambiguation and identification of author names [9], i.e., identifying the actual person an author name stands for and online data integration in federated search applications like DAFFODIL [4,16].

In a wider context, bibliographic metadata, which can be parsed from bibliographic references, play an important role for indexing and retrieval in digital object repositories like Greenstone[3] and Fedora.[4] In particular, the primary objects stored in such systems are not necessarily full-text indexable, for instance if they consist of (scanned) images or movie clips. This makes their metadata the sole searchable part, and thus indispensable for access. This scenario is

G. Sautter (✉) · K. Böhm
Karlsruhe Institute of Technology (KIT),
Am Fasanengarten 5, 76128 Karlsruhe, Germany
e-mail: sautter@ipd.uka.de

K. Böhm
e-mail: klemens.boehm@kit.edu

[1] http://citeseerx.ist.psu.edu/index.

[2] http://scholar.google.com.

[3] http://www.greenstone.org/.

[4] http://fedora-commons.org/.

Published online: 27 February 2014

common in digital preservation, rendering metadata in this field even more important. For preserved literary works and other books in particular, as available from Google Books[5], Internet Archive[6], Europeana[7], or the Biodiversity Heritage Library[8] [8], parsing a bibliographic reference extracted from some other work is a convenient way of obtaining the metadata, in contrast to entering it manually. Another application of bibliographic reference parsing is the extraction of detailed references from publications, as done by Citeseer [6] to determine which publication cites which other ones. This may take place for the purpose of indexing as well as to facilitate bibliometric analysis.

There are several essential terms we use throughout this paper: a (*bibliographic*) *reference string* is the unparsed reference as a whole, as found in the references section of this paper. The *data elements* are the individual attributes of the referenced work, e.g., author names and year of publication. The *reference style* is the arrangement of the data elements in a reference string, i.e., their order and their separating punctuation. A *reference list* corresponds to the references section of a publication; it comprises one or more bibliographic references formatted with the same style.

Several reference parsing algorithms [1,10,18] have been proposed in the recent past. There are two main categories: (1) pattern-based parsers like ParaCite [18] use regular expression patterns to match the individual elements of bibliographic references; these patterns are manually arranged in meta-patterns that reflect the order of the elements in reference style guides. (2) Training-based techniques like [1–3,7,10,11,13,24,26] use supervised learning to generate models of reference strings and the details they consist of from pre-parsed training examples.

However, both sorts of parsing algorithms mostly aim at the bibliographies of contemporary publications, which are highly standardized and follow one of only a few styles. The styles used for bibliographic references in the past are more manifold, so there is a lot more variation. This significantly increases the amount of training data required by training-based techniques to achieve good results. Furthermore, older references often make excessive use of abbreviations, which pose additional morphological challenges to pattern matching, and straightforward lookups in dictionaries and knowledge bases are not feasible. In consequence, ensuring data quality with the existing algorithms would require manual corrections of significant extent.

To improve parsing accuracy for the bibliographies of historic publications, this paper presents a new parsing algorithm named RefParse. RefParse integrates and generalizes approaches taken in previous works. In addition, it has several distinctive novel features: (1) it handles entire lists of references together. This facilitates exploiting the redundancy contained in such lists, on several levels. (2) It is truly independent of any particular reference style because it infers the style at hand at runtime. (3) It makes data quality control an integral part of the parsing process and learns from the manual corrections by users.

Technically, RefParse builds on both categories of parsing algorithms mentioned above, additionally incorporating manual correction and online learning. In particular, RefParse integrates four basic mechanisms: (1) RefParse uses regular expression patterns to match the individual elements of bibliographic references, similar to ParaCite, adding fuzzy dictionary lookups to recognize instances of data elements learned from previously parsed references, also covering abbreviated forms; (2) it represents the arrangement of the data elements in meta-patterns, again similarly to ParaCite, but dynamically infers the reference style in use instead of requiring it to be entered explicitly before deployment; (3) it optionally (in interactive mode) includes user interaction to improve parsing accuracy; and (4) after user approval, it learns concrete instances of specific data elements to recognize them later, namely the names of persons (authors and editors), periodicals, and publishers. The values of these elements are likely to reoccur in other reference lists from the same scientific domain that might be parsed in the future.

The basic ideas behind RefParse are inspired both by a thorough investigation of real-world data and by related work. In particular, RefParse combines previous work as follows: patterns, as used in [18], can reliably identify data elements with distinctive morphological properties, such as numeric elements, and these elements are helpful in delimiting other elements from one another. However, such delimitation tends to fail if the data elements relied upon are themselves ambiguous, e.g., in references whose title includes numbers. To resolve ambiguous cases, RefParse aligns possible arrangements of data elements across entire reference lists against one another, akin to [1,11], exploiting that all references follow the same style. By doing so, it can tell, for instance, that a four-digit number cannot be the year of publication if it occurs in a position where a majority of the references do not include numbers. Finally, RefParse uses a hybrid of knowledge-based techniques [2,3] and hidden Markov models [10,24,26] to extract names of periodicals and publishers.

In more detail, RefParse works as follows: first, it finds the data elements that are easy to identify based on morphological clues (Steps 1 and 2 below), so-called *base elements*. Second, it disambiguates and verifies the positions of the base elements in the reference style at hand by comparing the references in a list to each other (Steps 2 and 3). Third, it uses the base elements as auxiliary delimiters to extract the

[5] http://books.google.com.

[6] http://archive.org.

[7] http://europeana.eu.

[8] http://www.biodiversitylibrary.org/.

remaining data elements (Steps 4–6). In interactive mode, i.e., with users correcting the parsing results, there is a fourth stage in which RefParse prompts users for corrections and then learns from their input (Steps 7 and 8). We explain all the steps in detail in Sect. 4; here is an overview:

1. Identify possible values of the base elements, e.g., the year of publication, the pagination, and possible lists of author names. Besides morphological clues, also use dictionaries of data element values learned in the past, e.g., for author names.
2. Decide on the style of the author lists, e.g., the punctuation scheme used for individual names and the order of first name and last name. This works by means of a majority vote on the position, punctuation scheme, and name part order of all possible author lists. Consider only author lists that match the inferred style in subsequent steps.
3. Generate all possible orders of the candidate data elements identified in Step 1, also observing punctuation marks separating them. Then derive the actual order of data elements by means of a majority vote. Afterward, extract the actual values of the base elements based on the identified element order.
4. Identify possible embedded references to volumes, consisting of a volume title and potentially a list of editor names. Parse these embedded references recursively.
5. Identify the names of periodicals, titles of proceedings, and names and locations of publishers, whichever are present in the individual references.
6. Identify the titles of referenced works. Titles are the last data element identified because they exhibit the least regularity.
7. In interactive mode, have one or more users double check the parsing result, to ensure data quality.
8. Remember data element values that are likely to reoccur in other references, e.g., names of persons (authors and editors), names of periodicals, and names and locations of publishers.

This strict bottom-up approach renders RefParse mostly immune to erroneous or incomplete references. Namely, it works only with the data elements it actually finds candidate values for. In particular, this means that RefParse does not make any assumptions as to which data elements *should* be present. The latter is generally impossible to tell without knowing which kind of work a given reference string refers to, and RefParse does not make any assumptions in this latter regard.

RefParse works on plain text reference strings, which it tokenizes and dehyphenates internally before the actual parsing begins. We are aware that formatted text, like RTF or HTML, might contain further clues as to where specific data elements start or end, namely changes in the font style or even the font proper. However, RefParse does not exploit such clues for the sake of generality; in addition, the semantics of layout changes can vary considerably.

Our evaluation on a large body of reference lists from the biosystematics domain from the last 200 years shows that RefParse performs comparably to or better than previous approaches for contemporary reference lists, and that it also works well with historic ones. The latter is due to the high degree of flexibility RefParse has with regard to the reference style. Its results are already better in fully automated mode, i.e., without user interaction and runtime learning, achieving 94.3 % word accuracy on average. When users correct the parsed references, RefParse can extend its dictionaries at runtime. This increases word accuracy to 96.6 % in our experiments, meaning that users have to make corrections for only <4 % of the words. The latter makes RefParse very promising for community-backed construction of comprehensive bibliographies for entire scientific domains, like the bibliography of life effort currently undertaken as part of the ViBRANT project [25].

This paper extends a previous, shorter paper [22] at the content level in two ways: first, the solution described here is more refined than the earlier one. For instance, we now investigate a dictionary-related phenomenon we had observed in the short version and devise countermeasures. Second, this paper features a description and evaluation of RefParse in interactive mode, an extension that adds online learning capabilities and is particularly important in real-world applications.

Paper Outline Section 2 describes the basic types and elements of bibliographic references and presents problematic cases. Section 3 discusses related work. Section 4 presents our generic approach to bibliographic reference parsing, which we systematically evaluate in Sect. 5. Section 6 concludes.

2 Basics of references

This section describes the basics of bibliographic references in our context. From a sample of the documents we strive to extract detailed references from, we have compiled the types of works references can point to, the data elements they comprise, and the frequency of the data elements. To ensure the generality of our data model, our lists of data elements and types of work closely follow the widely used and largely universal BibTeX [19]: the list of types overlaps considerably with the one of BibTeX, but is not a subset of it; because they often comprise the same or nearly equivalent data elements in practice, we conflate several types. On the other hand, BibTeX does not cover references to volumes of periodicals. The list of data elements is a subset of the one of BibTeX. Again, we conflate several elements for practical reasons. This is because they are harder to distinguish morphologically than

based on the other data elements they occur in the same reference with: for instance, while both names of periodicals and names of publishers are generic proper names, the former usually occur together with a volume number, and the latter without. On the other hand, RefParse can handle a broader variety of data element values (especially person names) than covered by the BibTeX guidelines; this has turned out necessary to handle existing data.

Table 1 lists the reference types and data elements RefParse works with, together with examples and a mapping to BibTeX. In addition, the remainder of this section discusses some of the data elements and their characteristics and peculiarities in more detail.

The *author names* identify the creators of a referenced work. Bibliographic references often give a combination of a first name or initial and a last name (the same for *editor names*, if present). Depending on the country a person comes from, last names in particular can be rather complex, e.g., Spanish ones, not to mention double names. Depending on the reference style, the first and last names can occur in multiple orders and with various punctuation schemes, and the order of occurrence can change within a reference. In particular, the latter happens if the first author is given the last name first, and all subsequent authors the first name first.

Author names can further include several optional parts besides the first and last name: middle initials (e.g., "Alex *U.* Thor"), leading initials (e.g., "*S.* Edward Cond"), or middle names, affixes that refer to a generation (e.g., "Jr." or "IV.") or a noble title (e.g., "Duke of York"), or infixes like "van" or "von der". Depending on the reference style, these optional parts may occur in several positions and with various punctuation schemes. See Tables 1 and 2 for additional examples.

The author names are present in almost all bibliographic references. However, there are a few noteworthy exceptions: (1) they can be omitted if they are the same as the ones of the work whose reference precedes the given one in a reference list; (2) web pages may not have explicit authors in a bibliographic sense, or they are unknown, as is the case with Wikipedia articles; (3) references to publications like official reports of government agencies rarely include author names. In these cases, we have to avoid mistaking some other part of the reference string for author names.

References to journal volumes or articles almost always comprise the *name of the periodical*. These names usually rather uniquely identify the periodical by themselves, i.e., without requiring the actual publisher of the periodical in addition. This holds at least for major periodicals and at least within individual scientific domains. The names of periodicals issued by (scientific) institutions often include the name of these institutions or parts thereof. Periodical names are proper names and thus usually in title case. They can be very short, down to single words (e.g., "Nature" or "Science"), but they also can comprise eight or more words (e.g., "Transactions of the Royal Entomological Society of London"). Especially the latter periodical names are often abbreviated, with stop words like prepositions and determiners omitted (e.g., "Trans. Roy. Ent. Soc. Lond."). Apart from dots in abbreviations, periodical names close to never include punctuation marks.

In references to publications that consist of multiple physical volumes, *part designators* indicate in which of them to find the referenced work. They are highly common for periodicals, but can occur for books and manuscripts as well.

There is no consistent or even compulsory designator scheme for individual physical volumes of periodicals. Here are the most common variants (Personal Correspondence with Terry Catapano, Columbia University Library, New York, NY, USA):

- Physical volumes have consecutive volume or issue numbers.
- Physical volumes are bundled into logical ones (e.g., by year of publication), distinguished by consecutive issue numbers that can run across logical volumes or start over at 1 with each new volume. Especially in the latter case, there can additionally be a numero that increases with each physical volume across logical volumes.
- Physical volumes are numbered like in the previous bulletin, but with the roles of volume and issue numbers swapped.

Volume, issue, or numero designators are usually numbers, mostly Arabic, less commonly Roman ones. Preceding respective labels like "vol.", "issue", or "no." are not uncommon. If two part designators are given, most commonly volume and issue, they often occur without labels, next to one another, with the issue designator in brackets.

There can be even more subdivisions in periodicals, e.g., series. Further, books and manuscripts can the printed in multiple volumes or fascicles, respectively.

3 Related work

The problem of parsing individual data elements out of bibliographic reference strings has received considerable attention in the recent past. The approaches fall into two general categories: (1) pattern-based parsers like [18], and (2) parsers generated from training data by means of machine learning. The latter approaches again fall into three subcategories: (2a) classic machine learning-based parsers using statistical models [7,10,13,24], (2b) parsers relying on knowledge bases created from training data [2,3], and (2c) parsers based on sequence alignment algorithms [1,11], which were originally developed for protein sequence alignment.

Table 1 Types of works and data elements in respective references

Data element (BibTeX names)	Examples/ morphology or remark	Occurrence in references to (BibTeX names)						
		Book (book, phdthesis, masterthesis)	Book chapter (inbook, incollection)	Journal volume (none)	Journal article (article)	Proceedings (proceedings)	Proceedings paper (inproceedings)	URL (none)
Author (author)	A. U. van Thor, Jr., Thor, A. U. van, Jr., Thor, Jr., A. U. van, van Thor, A. U. Jr., van Thor, Jr. A. U.	Yes (can also be editors placed in author position)	Yes	Yes (can also be editors)	Yes	Yes (often editors, but placed in author position)	Yes	If known
Title (title)	No regularities, can contain anything, have any length	Yes	Yes (can be substituted with "Untitled")	Special issues only	Yes	Yes	Yes	Yes
Year of publication (year)	Four-digit Arabic number from [1600, 2100]	Yes	Yes	Yes	Yes	Yes	Yes	If fix, known
Publisher name (publisher, institution, school)	Morgan Kaufman, Harvard Univ. Press, Univ. of Florida	Yes (if not unknown)		Rare	Rare	Rare	Rare	No
Place of publication (address)	Any city, possibly with state and/or country	Yes [if not in name of publisher (institution)]		Rare	Rare	Frequent (can also be the conference location)		No
Periodical name (journal)	Nature, Science, VLDB Journal, Jour. Nat. Hist.	No	No	Yes	Yes	If published as special issue of journal	If proceedings published as special issue of journal	No
Part designator (volume)	(Mostly Arabic) number, possibly with label	Not in this sense (see text for details)		Yes	Yes			No
Pagination (pages)	Two (mostly Arabic) numbers separated by dash	No	Yes	No	Yes	No	Frequent	No
Volume title (booktitle)	See title	No	Yes	No	Special issues only	No	Yes	No
Editor (editor)	See author	No	Frequent	No		No	Frequent	No
Digital identifier (url)	Any URL or DOI	Rare	Rare	Rare	Rare	Rare	Rare	Yes
Content information (note)	216 pp., 17 figures, with illustrations	Frequent	No	No	No	No	No	No

Table 2 The name *Alex U. Thor* in different styles

Name part order	First name style	Name style ID	Examples
Last name last	Written out	FL-N	Alex U. Thor Alex Thor Alex U. THOR Alex THOR
Last name first	Written out	LF-N	Thor, Alex U. Thor, Alex THOR, Alex U. THOR, Alex
last name last	Initials	FL-I	A. U. Thor AU Thor A. U. THOR AU THOR
Last name first	Initials	LF-I	Thor, A. U. Thor, AU THOR, A. U. THOR, AU Thor AU THOR AU

Pattern-based parsers like ParaCite [18] rely on morphological clues to identify the individual data elements in bibliographic reference strings. They represent the reference style as meta-patterns concatenated from the patterns for the individual elements. Patterns are well suited to identify certain data elements based on morphological clues, e.g., years of publication, person names (authors, editors), part designators, or names of periodicals and publishers. However, the meta-patterns have to be created manually at configuration time. This renders the approach incapable of self-adapting to reference styles not covered by the training data. In consequence, RefParse uses patterns only for the identification of individual data elements whose instances have a distinctive morphological structure. It uses other means to identify the remaining data elements, which we describe in Sect. 4, and it infers the reference style automatically at runtime.

Classic machine learning-based parsers use statistical models to identify the individual data elements in bibliographic reference strings. There have been several approaches using hidden Markov models [10,24,26], as well as finite state transducers [13]; ParsCit [7] in turn uses conditional random fields. The major weakness of all these approaches is the need for parsed training data for supervised learning, which must cover all reference styles the parser is to process later on. While such data are readily available in sufficient amounts for many contemporary reference styles, this is not the case for the multitude of older styles used in historic publications, where obtaining a sufficient number of representative training examples can become a major bottleneck and cost factor.

Knowledge-based parsers [2,3] follow a somewhat different approach: instead of statistical models, they create knowledge bases from their training data. FLUX-CiM [2] divides reference strings into blocks at punctuation marks and then classifies these blocks by means of lookups in a knowledge base. Cortez et al. call it unsupervised because the data they populate the knowledge base with is readily available for their test sets. However, FLUX-CiM actually does require parsed training data. This makes it supervised learning, no matter whether that data is readily available or not, and in addition the training data have to be from the scientific domain as the references to parse. The latter is to make sure that at least some percentage of the knowledge base lookups actually results in a match. Respective data cannot be assumed to be readily available for the reference lists found in historic publications. In addition, FLUX-CiM achieves its lowest accuracy for the most important data elements, namely the title of the referenced work and the name of the periodical it has been published in. Further, the evaluation by Cortez et al. only covers two contemporary reference styles and uses homogeneous test sets exclusively consisting of references to articles published in periodicals and conference proceedings. It is unclear how FLUX-CiM would perform on heterogeneous reference lists that also comprise references to other types of publications listed in Sect. 2: references to books and book chapters in particular are very common in life science publications. An additional problem with historic reference lists is that they often include abbreviated data elements, which (1) have internal dots that interfere with the blocking and (2) make knowledge-base lookups highly challenging. The latter means that abbreviations need to be matched against knowledge-base entries, which are usually not abbreviated. Approaches that focus on contemporary reference lists do not address this problem. INFOMAP [3] creates a hierarchical, domain-specific template base from its training data. In the parsing process, it matches the reference strings against these templates. These researchers have focused exclusively on references to articles published in periodicals in their evaluation [3]. As mentioned before, however, reference lists of life science publications often are more diverse. Again, the performance on heterogeneous reference lists like the bibliographies of real-world publications is unclear.

Despite its drawbacks when used as the main basis for parsing, the knowledge-based approach is promising for data elements whose instances are likely to occur multiple times. In particular, RefParse can use dictionaries of person names, names of periodicals, and publisher names and locations, facilitating recognition of names seen previously. Further-

more, RefParse can use statistics on the probabilities of words to be part of periodical or publisher names to find the boundary between either of the latter and the publication title. This is particularly helpful with references whose periodical or publisher name is not in the respective dictionary as a whole, but includes words that are frequent in such names. However, RefParse does not depend on these dictionaries and statistics, and thus does not require training data; it uses them solely as a means of improving parsing accuracy. In interactive mode, i.e., when users double check and correct parsing results, RefParse extends both the dictionaries and the statistics at runtime. If available, readily compiled dictionaries from external sources can be integrated as well. To handle abbreviations, dictionary lookups use a respective fuzzy matching mechanism. Lookups in the statistics on individual words in turn do not do so. This is because this would be too ambiguous and thus too likely to incur errors.

Sequence alignment based parsers [1,11] employ algorithms originally developed for protein sequences; the classes of individual words, numbers, and punctuation marks take the place of the base pairs. They rely on a large base of templates generated by means of supervised learning. Alignment matches the reference strings to parse against these templates and extracts the most similar one. The latter then specifies how to split the reference strings into the individual data elements. As with the machine learning techniques discussed above, the main drawback of sequence alignment parsers is their need for parsed training data. It has to include examples for all possible reference styles to make sure the parser has a respective template available. A second drawback is that data elements with widely varying instances, such as the title, are highly challenging for alignment. This is because they need to match a template from the training set. A third weakness is that only the best matching template is considered for each reference string; when parsing a list of references that follow the same style, it seems more promising to select the template that is the best match from a global point of view, i.e., for all references in synopsis.

RefParse uses the alignment approach to identify the position and delimiting punctuation of data elements that occur in many reference strings within a list, and whose possible instances can be identified based on morphological clues. In particular, these elements are the list of author names, the pagination, the part designators, and the year of publication. Data elements whose instances vary widely across references, e.g., titles, are not subject to the alignment. RefParse extracts them later on and thereby avoids having to deal with their variety at this stage. RefParse then generates all possible arrangements for all identified data elements from all references in a list and then selects the globally best one by means of mutual alignment. The underlying assumption is that all references in a list follow the same (to-identify) style. This mechanism is different from [1,11] in that it does not match the alignments against a pre-existing set of templates learned in a training phase, but against one another. In addition, it considers multiple arrangements per reference string. As a consequence, RefParse is truly independent of any training data or particular reference style. Furthermore, RefParse does not depend on matching references against previously learned templates, where missing data elements can prevent a match and thus incur errors [24]. It works strictly bottom-up, starting from the data elements it finds candidate values for, and without making any assumption which data elements *should* be present. The latter naturally varies depending on the kind of work a reference refers to. Not making any assumption in that regard inherently also renders RefParse mostly immune to erroneous or incomplete references.

4 The refparse algorithm

This section explains the RefParse algorithm in detail. First, we introduce three running examples to illustrate our explanations. Second, we explain the basic design of RefParse and the rationale behind it, and finally we go through the parsing process step by step.

There is one *basic assumption* behind RefParse: all bibliographic references in the respective section of a publication are formatted in the same way. This applies both to the formatting of lists of person (author/editor) names and to the order of the individual data elements in the references. This assumption is reasonable: a thorough investigation of the Plazi Corpus (a corpus of over 1,000 real-word documents from the last 200 years, see Sect. 5.1) has not revealed any counterexamples.

The design of RefParse is the result of a thorough investigation of both real-world bibliographic reference data and the specific strengths and weaknesses of previous approaches to bibliographic reference parsing. RefParse is geared toward maximum result accuracy in real-world application scenarios. Our evaluation and experimental comparison to related work (cf. Sect. 5) shows that RefParse does meet this design goal.

4.1 Running examples

This section introduces three running examples we will use to illustrate the explanations to come. Each one represents a class of cases that might even appear constructed at first glance. However, they are too frequent in practice to be ignored.

Example 1 is a reference to a single page journal article; it illustrates the ambiguity of numerical elements and helps explain how RefParse disambiguates them:

Thor, AU, Cond, SE. This is the article title. The Particular Journal 1976 (1998): 1987.

In particular, each of *1976*, *1988*, and *1987* could be the year of publication, but one actually is the volume number and one the pagination. It is impossible to figure out which one is which without making any assumptions regarding the order of these data elements or their intermediate punctuation. It is a core design feature of RefParse to not make such assumptions to achieve maximum generality.

Example 2 is a reference to a conference paper; it illustrates the occurrence of numbers that are not numerical elements, but parts of others (the proceedings title, in this case), and how RefParse avoids mistaking them for numerical elements.

Thor, AU. The paper title. In: Proceedings of the 19th International Conference on Things and Stuff (ICTS 2012), The Venue, 2012.

Example 3 is a reference to a book that includes a DOI some parts of which resemble a pagination element; it illustrates the pitfalls encountered in references to contemporary publications and how RefParse avoids falling into them.

*Thor, AU. The book title. Somewhere: The Publisher, 2011. doi:*10.1000/182-184.

We will refer to any of these examples where it eases understanding.

4.2 Basic design

The basic ideas behind RefParse are inspired both by thorough investigation of real-world bibliographic references and by the experiences and evaluation results reported in related work. In particular, some data elements exhibit a highly distinctive structure and thus are easy to identify by means of patterns. Very high field accuracy values (cf. Sect. 5.1) for respective elements corroborate this assumption. The most prominent element with this characteristic probably is the year of publication, a four-digit Arabic number. In addition, it occurs in many references. This makes it a reliable means of delimiting other data elements whose structure is less distinctive, e.g., author names and title.

However, investigation of real-world data also shows that such a delimitation scheme often causes trouble with references that comprise more than one four-digit number, e.g., as a part designator or as part of the title or volume title, as in Example 2. The same applies to other numeric elements as well, namely pagination and part designators. The actual semantics of a number, i.e., which data element it belongs to, in general is only clear in its context, and the context may be confusing as well occasionally.

To illustrate the latter, look at Example 1: it is impossible to reliably tell even for a human which of the three numbers is the volume number, which one the year or publication, and which one the page number. Their order gives some clues, but they are insufficient to be sure.

While this ambiguity is not unlikely for individual references, it rarely occurs across an entire reference list. Because all references in a list follow the same style, their data elements have the same position. Thus, comparing possible arrangements of data elements across a reference list can resolve ambiguous cases.

Now, suppose the reference from Example 1 occurs in a bibliography together with the following one:

Cond, SE. Some other article title. Another Journal 16 (1995): 23–28.

In this latter reference, it is rather obvious that 16 is the volume number, 1995 the year of publication, and 23–28 the pagination. This clarifies the order of the numerical elements and thus facilitates understanding the semantics of the numbers in Example 1. We refer to this clarifying reference as Example 1a in the following.

Capitalizing on this observation, RefParse works as follows: first, find the data elements that are easy to identify based on morphological clues (Steps 1 and 2 below), referred to as the *base elements*. Second, disambiguate and verify the positions of the base elements in the reference style at hand, exploiting that all references in a list follow the same style (Steps 2 and 3). Third, use the base elements as auxiliary delimiters to extract the remaining data elements (Steps 4–6).

RefParse can work in two modes: in *fully automated mode* (Steps 1–6 only, Steps 7 and 8 are omitted), it works autonomously, like any other reference parsing system; in *interactive mode* (Steps 1–8), it has users double check and correct parsing results to ensure data quality in practical deployment; it furthermore learns from user corrections.

4.3 Parsing steps

RefParse works in eight steps, as visualized in Fig. 1. It processes all bibliographic references from the bibliography section of a publication together. The references go through the parsing process in parallel, i.e., one step runs on all references before proceeding to the next step. The first six steps run in both fully automated and interactive mode, and the last two steps in interactive mode only.

In the following explanations, the term *references* generally refers to the references that make up the bibliography section of a publication; *preceding reference* means the one preceding the one under investigation in the reference list, i.e., the one right above it. The parsing of Example 1 in the context of a similarly formatted list of references (represented by Example 1a) illustrates how RefParse works.

Step 1: Base element extraction This step identifies possible instances of the base elements, i.e., years of publication,

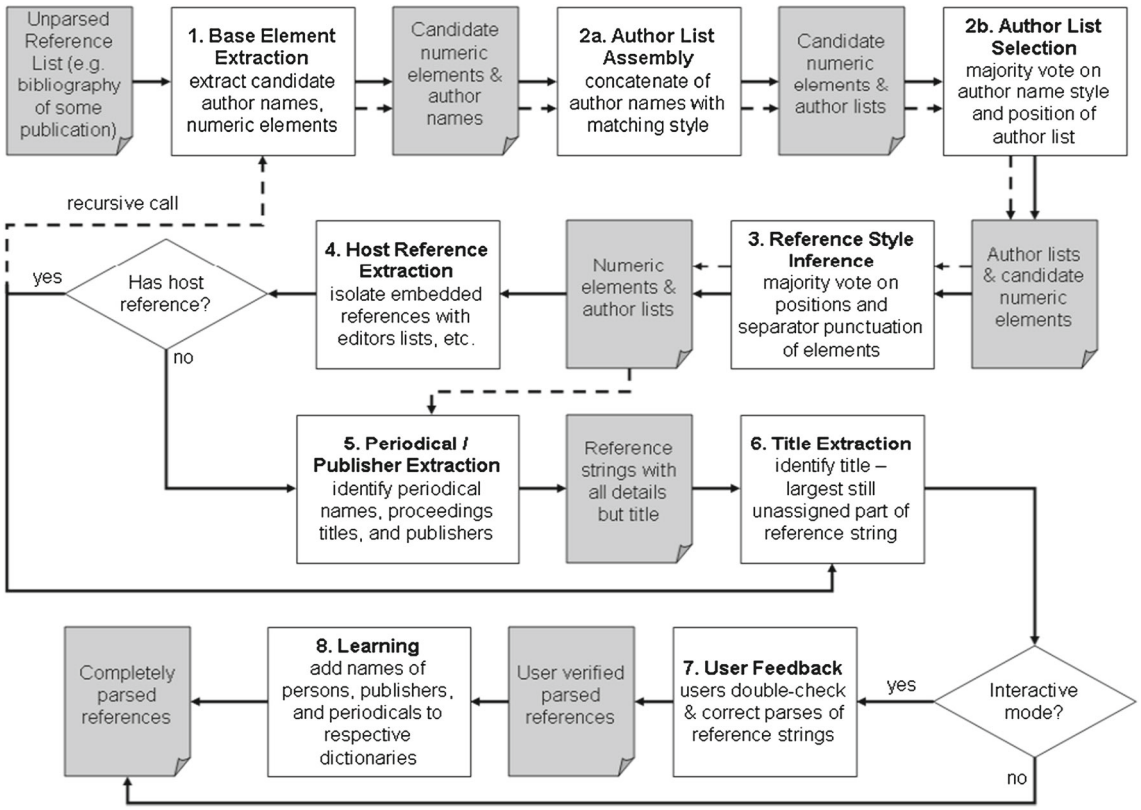

Fig. 1 The RefParse algorithm

part designators, and pagination, as well as possible author names. The rationale is that these data elements have a distinctive morphological structure and at the same time occur in many references. The distinctive structure allows to reliably identify possible data element values by means of patterns. The high probability of occurrence ensures that there are several examples of the actual data element positions in the reference style in use. This is essential for the majority vote used in Step 3, to identify the positions and deal with ambiguities. All patterns used in this step are designed to match any possible value of the data elements they are intended to match, to avoid misses. The ambiguous cases occasionally resulting from this approach are tolerable in this stage; Step 3 will deal with them. Note that the patterns exclusively match individual data elements, not the reference string as a whole.

RefParse relies on patterns for the numerical elements: years of publication are four-digit Arabic numbers; paginations are either single Arabic numbers, or pairs of them with a dash in between; part designators are numbers, both Arabic and Roman, and sometimes single capital letters. If the interpretation of a number is ambiguous, all possible interpretations are considered. To illustrate, a four-digit number becomes a candidate for any of the numeric elements; numbers with preceding part designator labels like "vol." or "no." become candidates only for this data element.

Even though the numerical base elements are easy to extract because of their structure, there are a few ambiguous cases to filter. Namely, we have to avoid finding candidate values for data elements that do not actually exist in a given reference. Numerical parts of digital identifiers are a particular problem here, like the DOI in Example 3: the part *182–184* perfectly resembles a value of the pagination element. Ordinal numbers and years that belong to book titles, the titles of proceedings volumes in particular, are problematic as well: the *19* in Example 2 is quite close to where the volume number would be in a reference to a journal article, but it is a part of the title of a proceedings volume in this case.

To avoid this kind of ambiguity, RefParse extracts URLs, digital identifiers, ordinal numbers, and conference abbreviations (like *ICTS 2012* in Example 2) even before the candidate values for the numerical base elements. It then sorts out any candidate values for the latter that lie inside any of the former. Because the structure of URLs, digital identifiers, and ordinal numbers is even more distinctive than that of plain numbers sought for in isolation, this filtering is extremely unlikely to result in missing any actual numerical base elements, while preventing many false positives.

For author names, RefParse uses several different patterns, each reflecting an individual formatting style. The space of styles has two dimensions: the patterns distinguish the order

of first and last name, and whether first names are written out or generally given as initials. It can happen that parts of the references match several name styles: the pattern matching *Alex THOR* in Table 2, for instance, will also match *Thor AU* because both strings have the same morphological structure. *Alex THOR* could also be short for *Tom H. O. R. Alex*, with *Alex* being the last name, and *AU* could be the last name in *Thor AU*. To avoid errors, parts matched by patterns for several name styles become candidates for all of them. The patterns are designed to match both first names and last names consisting of multiple parts, with intermediate infixes, hyphenated double names, etc. Table 3 provides examples of the candidate names extracted from a given author list. Furthermore, there are patterns for author repetition marks (usually one or more dashes) that indicate that the authors are the same ones as in the preceding reference, and for the special string "et al.". The distinction of styles helps in Step 2, namely to only concatenate author names with matching styles.

To reduce computational complexity, our implementation of RefParse does not use patterns that extract potential author names as a whole in a single pass. It rather uses patterns that extract the individual parts of author names discussed above. Afterward, meta-patterns combine these parts to full potential author names. Figure 2 explains the patterns for the first and last names in detail; the ones for initials and blocks thereof are trivial. Infixes and affixes are extracted by means of dictionaries rather than full-fledged patterns.

Before assembling the individual parts to complete potential author names, RefParse performs several intermediate aggregations to obtain further candidate parts: In particular, it aggregates (1) potential first names with adjacent ones, as well as with initials, to form additional first names, (2) potential last names with adjacent ones, as well as with adjacent infixes, to form additional last names, and (3) potential last names with adjacent affixes to form yet additional last names.

The meta-patterns that finally assemble the individual author name parts to potential author names correspond to the author name styles listed in Table 2. For some of the latter, there are several meta-patterns. This is to reduce the complexity and still cover all possible positions of infixes and affixes.

The meta-patterns for author names in the last-name-last styles are relatively straightforward:

- <firstname> <infix>? <lastname>(\, ? <affix>)?
- <initials> <infix>? <lastname>(\,? <affix>)?

The ones for author names in the last-name-first styles have to cover more variation, especially regarding the placement of infixes and affixes; see Table 1 for examples of placements of the latter:

- <infix>? <lastname>(\, ? <affix>)?\, <firstname>
- <infix>? <lastname>\, <firstname>(\, ? <affix>)?
- <lastname>\, <firstname>(\, ? <infix>)? (\, ? <affix>)?
- <lastname>(\, ? <affix>)?\, <firstname> (\, ? <infix>)?
- <infix>? <lastname>(\, ? <affix>)?\, <initials>

Table 3 The names *Alex U. Thor*, *Steve E. Cond*, and *Tom Hird* listed in different styles

List style	Example	Names found in Step 1 (grouped by style)		
		LF-I	LF-N	FL-I
LF-I, FL-I	Thor, AU, SE Cond, T Hird	Thor, AU AU, SE Cond, T	AU, SE Cond Cond, T Hird	SE Cond T Hird
LF-I	Thor, AU, Cond, SE, Hird, T	Thor, AU Cond, SE Hird, T	AU, Cond SE, Hird	–
LF-I, FL-I	Thor, AU, SE Cond, and T Hird	Thor, AU AU, SE	AU, SE Cond	SE Cond T Hird
LF-I	Thor, AU, Cond, SE, and Hird, T	Thor, AU Cond, SE Hird, T	AU, Cond SE, Hird	–

RefParse uses the following pattern to extract potential last names:

```
(([A-Z][a-z]+)|([A-Za-z][a-z]*\'))?[A-Z]([a-z]+|[A-Z]+)(\-(([A-Z][a-z]+)|([A-Za-z][a-z]*\'))?[A-Z]([a-z]+|[A-Z]+))?
```

To explain, the individual parts of this pattern have the following functions:

`(`	
`([A-Z][a-z]+)`	a prefix like *Mc*, *Mac*, etc.
`|`	
`([A-Za-z][a-z]*\')`	a prefix with an apostrophe, like *O'*, *l'*, *d'*, *del'*, etc.
`)?`	
`[A-Z]([a-z]+|[A-Z]+)`	the main part of the last name, capitalized or in all-caps
`(\-(([A-Z][a-z]+)|([A-Za-z][a-z]*\'))?[A-Z]([a-z]+|[A-Z]+))?`	the same, with an initial dash, to capture double names

First names are simpler; we do not need to consider the apostrophe prefixes, and no all-caps spelling, either:

```
([A-Z][a-z]+)?[A-Z][a-z]+(\-([A-Z][a-z]+)?[A-Z][a-z]+)?
```

Fig. 2 Author name extraction patterns explained

- <infix>? <lastname>\, ?<initials>(\, ? <affix>)?
- <lastname>\, ?<initials? <initials>(\, ? <infix>)?

In addition to the patterns and meta-patterns, RefParse can use a dictionary of person names to look up the candidate author names. While increasing accuracy, however, this additional source of knowledge is not required for RefParse to work.

After Step 1, RefParse has the following candidate combinations for the reference from Example 1 (a few are omitted to retain readability):

- For the author names:
  Last name first, initials: *Thor, AU; Cond, SE*
  First name first, initials: *SE. This*
  First name first, full first name: *The Particular Journal*

- For the numerical data elements:
  Year: *1976 1998 1987*
  Part designator: *1976 1998 1987*
  Pagination: *1976 1998 1987*

At the same time, RefParse gets the following for Example 1a:

- For the author names:
  Last name first, initials: *Cond, SE*
  First name first, initials: *SE. Some*
  First name first, full first name: *Another Journal*

- For the numerical data elements:
  Year: 1995
  Part designators: *16 1995 23 28*
  Pagination: *16 1995 23–28*

Step 2: Author list assembly and selection This step identifies the list of author names in each reference. It works in two phases.

First, the *assembly phase* creates all possible lists of author names for each reference from the possible names found in Step 1. It concatenates only names with matching style, i.e., with the same order of first and last name and the same way of giving the first name. There is one exception with regard to the order of the first and last name: it may be the last name first for the first name of a list and first name first for all subsequent ones. Between two concatenated names, there has to be a separating punctuation mark and/or a conjunction. Author repetition marks (see above) can only be the first part in an author list and "et al." can only be the last one.

Second, the *selection phase* chooses the most likely author list for each reference from the possible author lists created in the assembly phase. Its underlying assumption is that reference lists are consistent with regard to the style and positioning of the author lists, i.e., the style of the author list is the same in each reference in a list and it has a similar position in each one.

After the assembly phase, RefParse has the following candidate author lists for Examples 1 and 1a, respectively:

- Last name first, initials: *Thor, AU, Cond, SE*
  First name first, initials: *SE. This*
  First name first, full first name: *The Particular Journal*

- Last name first, initials: *Cond, SE*
  First name first, initials: *SE. Some*
  First name first, full first name: *Another Journal*

Note that the only difference to the state before the assembly phase is that consecutive author names of the same style are concatenated to form author lists.

In more detail, RefParse scores each author list style to assess how likely it is the one in use. It then chooses the style with the highest score. Figure 3 formalizes the scoring function in pseudo code. Informally, RefParse considers an author list style more likely to be the one used in a given reference list

1. the more references contain an author name list of that style, i.e., the more references it matches an author name list,
2. the less ambiguous the matches are;
3. the more potential author names it matches,
4. the smaller the gaps are between matched potential author names;
5. the more similar the distance is between potential author lists and the start or end of reference strings,
6. the more distinctive the style is.

Points 1–3 are intuitive. The motivation for Point 4 is the observation that author lists solely consist of author names with only specific punctuation marks and conjunctions between them. Point 5 is based on the observation that author lists are next to never located in the middle of reference strings. In particular, they are not located between title and periodical name or publisher, which are the other two detailed attributes whose values considerably vary in length. Thus, the distance of the author lists to one end of the reference strings in a reference list is relatively constant. It depends on the reference style, whether this is the start or the end. Point 6 reflects that the less distinctive an author list style is, the more likely it is to yield false positives. In the above illustrations based on Example 1, the periodical names

```
var rl :=                    the reference list being parsed
var als :=                   author list style being scored
var numRefs(rl) :=           number of references in rl

supportRefs(als, rl) :=      number of references als has at least one match in
support(als, rl) :=          total number of author list matches of als across all references in rl
supportNames(als, rl) :=     total number of author names contained in author lists counted by instCount(als, rl)
bridged(als, rl) :=          number of non-punctuation tokens between author names in lists matching als

firstNameStyle(als) :=       first name style als follows, namely I for initials or N for full first name
namePartOrder(als) :=        name part order als follows, namely FL for first name first or LF for last name first

SD(als, rl) :=               across all reference strings r ∈ rl the distribution of the distances (in tokens) between the start
                             of r and the first match of als in r (SD for start distance)
ED(als, rl) :=               across all reference strings r ∈ rl the distribution of the distances (in tokens) between the end
                             of r and the last match of als in r (ED for end distance)

// compute the score indicating how likely an author list style als is the one in use in a reference list rl
function scoreAuthorListStyle(als, rl) {
  if (supportRefs(als, rl) * 3 < numRefs(rl)) return 0;    // hard limit for Point 1 (matches in too few references)
  // compute base scores from distances to start and end of reference string (the scores are the higher (1) the less
     variation there is in the distance distribution, i.e., lower its entropy is, Point 5, and (2) the more references als
     has a match in, Point 1, H(...) is the Shannon entropy of the distance distributions)
  var startDistScore := 1 / (H(SD(als, rl)) + (1 / supportRefs(als, rl))²)
  var endDistScore := 1 / (H(ED(als, rl)) + (1 / supportRefs(als, rl))²)
  var score = max(startDistScore, endDistScore)        // use whichever base score is higher
  if (firstNameStyle(als) = N) score = score / 2       // Point 6 (penalize low distinctiveness of full first name)
  if (namePartOrder(als) = FL) score = score / 2       // Point 6 (penalize low distinctiveness of first name first)
  if (numRefs(rl) * 2 < support(als, rl)) score = score / 2 // Point 2 (penalize high ambiguity, > 2 matches per reference)
  score = score * supportNames(als, rl)                // Point 3 (reward high number of names)
  score = score / bridged(als, rl)                     // Point 4 (penalize large gaps between names)
  return score
}
```

Fig. 3 The scoring function for author list styles

are false positives for the *first name first, full first name* style. To prevent such errors, RefParse favors *last name first* over *first name first*. This is because the former has a more distinctive punctuation. It also favors *initials* over *full first name* for the same reason.

Finally, this step selects the actual author list for each reference from the candidates, based on compliance with the style just identified and its position in the reference. The latter helps to distinguish lists of author names from lists of editor names in the middle of a reference, which usually follow the same style as the author lists.

> In the running examples, RefParse selects *last name first, initials* as the author name style. This is because (1) this style has the most potential author names, namely three, *Thor, AU* and *Cond, SE* in Example 1 and *Cond, SE* in Example 1a, while the other styles only have two potential author names, and (2) yields the author lists that are in total closest to the reference boundaries, namely the starts of the reference strings in this case. The rationale is that in the vast majority of reference styles, the author lists are located very close to the beginning of reference strings, preceded by at most a reference number (like in this paper) and the year of publication. In the remaining reference styles, the author list is toward the end of the reference strings, succeeded at most by the year of publication. We have not found any style placing the author list in the middle.

Step 3: Reference style inference This step identifies the way the reference style in use arranges the numerical data elements identified in Step 1 and the author lists identified in Step 2. This includes both the positions of these elements and their separating punctuation. This again works in two phases:

The *assembly phase* generates all possible arrangements of the data elements identified so far, i.e., numerical data elements and author lists. A four-digit number, for instance, is (1) a possible year of publication, (2) a possible part designator, e.g., a volume number of a periodical, or (3) neither of the two. The latter reflects the fact that numbers can also

occur as part of publication titles, for instance. This phase also considers that some data element might not be present in a reference, even though Step 1 has found a candidate instance. Thus, it also generates additional arrangements with one or more such elements left out. The elements this applies to are the ones whose values are numbers or number ranges, namely year of publication, part designators, and pagination.

This bottom-up approach to reference style inference renders RefParse naturally immune to missing data elements, e.g., a reference to a journal article lacking the pagination. RefParse simply works with the data elements it finds candidates values for. From another angle, at this stage of parsing there is no knowledge or assumption regarding the type of work a reference refers to, and thus no knowledge about which data elements *should* be present if the reference to be complete. Hence, missing data elements do not affect RefParse at this stage.

The *selection phase* chooses the most likely arrangement of data elements for each reference. It scores all candidate arrangements generated in the previous phase based on multiple criteria. Figure 4 formalizes the respective scoring function in pseudo code. Informally, RefParse considers an arrangement and punctuation scheme to represent the reference style in use in a given reference list that is more likely:

1. the more data elements it contains,
2. the more references it is possible for, both regarding data element positions and inter-element punctuation; and
3. the closer the data elements stand together, the larger the individual chunks of the reference string are that remain unassigned.

The idea behind Point 1 is to favor candidate arrangements covering many data elements. Point 2 exploits that all references in a list follow the same (if unknown) style, so the instances of their data elements occur in similar positions within the references. It further considers that punctuation between individual detail attributes is an integral part of reference styles, and thus matching punctuation is a good indicator for a plausible arrangement of detail attributes. The motivation for Point 3 is twofold: (a) aside from reference strings

```
var dea :=                  the data element arrangement to score
abst(dea) :=                an abstraction of dea, coming in three variants:
  punct(dea) :=             detailed abstraction, representing order of data elements, unassigned chunks between, and
                            punctuation around them
  summary(dea) :=           medium abstraction, representing order of data elements and unassigned chunks between them
  order(dea) :=             coarse abstraction, representing only order of data elements
maxUnassigned(dea) :=       number of tokens in the longest unassigned chunk
DEAS :=                     set of all data element arrangements generated for the reference list
elementCount(dea) :=        the number of data elements in dea
frequency(abst(dea)) :=     the number of references the abstracted arrangement occurs for
subsumes(deaBig, deaSmall) :=   true if deaBig contains at least all the elements in deaSmall, and the elements are
                                in the same order in both, ignoring additional elements in deaBig

function scoreFuzzy(dea, abst()) {
  var sf = 0;
  for (cDea in DEAS) {
    var csf = 0;
    if (abst(dea) = abst(cDea)
      cfs = elementCount(dea)³ * frequency(abst(dea))        // Points 1 and 2 (reward exact match higher than fuzzy one)
    else if (subsumes(cDea, dea))
      cfs = elementCount(dea)² * frequency(abst(dea))        // Points 1 and 2 (still reward fuzzy match)
    sf = max(sf, csf)
  }
  return sf
}
function scoreDataElementArrangement(dea) {
  var score = 0
  score += scoreFuzzy(dea, punct())
  score += scoreFuzzy(dea, summary())
  score += scoreFuzzy(dea, order())
  score += maxUnassigned(dea)                                // Point 3
  return score
}
```

Fig. 4 The scoring function for data element arrangements

with an embedded host reference, the only two remaining detail elements are title and periodical name or publisher; each of them forms a continuous chunk of the reference string. (b) In most reference styles the latter two are adjacent to one another and thus even form a single chunk at this stage of parsing.

Finally, this step extracts the actual instances of the data elements by matching the positions of the candidate instances against the selected arrangement.

In the running examples, all this would work in the following way: Example 1 does not provide any clues regarding the element order. In particular, all three numbers are suitable values for all three numerical data elements. RefParse generates candidate arrangements for all six permutations, as well as arrangements omitting the part designator and/or pagination. For the reference string from Example 1, RefParse generates the following arrangements:

1 <authorList>_<year>_ <partDesignator> _<pagination>
2 <authorList>_<year>_ <pagination> _<partDesignator>
3 <authorList>_<partDesignator>_ <year> _<pagination>
4 <authorList>_<partDesignator>_ <pagination> _<year>
5 <authorList>_<pagination>_ <year> _<partDesignator>
6 <authorList>_<pagination>_ <partDesignator> _<year>
7 <authorList>_<year>_ <partDesignator>
8 <authorList>_<partDesignator>_ <year>
9 <authorList>_<year>_ <pagination>
10 <authorList>_<pagination>_ <year>
11 <authorList>_<year>

The reference string from Example 1a, on the other hand, is a lot clearer, resulting in fewer candidate arrangements:

1 <authorList>_<partDesignator>_ <year> _<pagination>
2 <authorList>_<partDesignator>_ <year>
3 <authorList>_<year>_ <pagination>
4 <authorList>_<pagination>_ <year>
5 <authorList>_<year>

Because it scores the highest, RefParse then selects the arrangement 1 from the latter list (number 3 in the former one), namely:

<authorList>_<partDesignator>_<year>_<pagination>

With this inferred information, RefParse can now interpret the numbers in Example 1: 1976 is the part designator, 1998 the year of publication, and 1987 the page number the referenced work has been printed on. The numbers in Example 1a are unambiguous from the start: 1995 is the year of publication, 23–28 the pagination, and 16 the part designator. However, it might have been the case that one of the numbers does not belong to any of the numerical elements, but rather is part of the title or publisher name.

Step 4: Host reference extraction This step identifies possible embedded references to volumes that consist of multiple data elements themselves and thus require parsing. Table 4 provides examples of embedded references, one to a book and one a proceedings volume. This step mostly aims at references to books embedded in references to book chapters, but there also are other cases, e.g., special issues of periodicals. The most common case of embedded references is a list of editor names and a volume title, followed by the name of a publisher or a periodical. Typically, a preposition like "in" precedes the list of editor names, often separated with a colon.

RefParse handles the embedded references found in this step recursively, processing them through Steps 1–3, 5, and 6. The rationale is that embedded references follow the same style, just as the top-level references they are embedded in.

Note that technically any pair of a periodical name and a part designator also constitute a reference to a volume. However, such pairs do not exhibit the complexity found in embedded references to books and thus do not require recursive handling.

In neither of Examples 1 and 1a, RefParse finds any indicator of a host reference, like a list of editors or the indicator string *In:*, so it proceeds with Step 5.

Step 5: Periodical and publisher extraction In the references Step 4 has not found an embedded reference; this step identifies the names of periodicals, the titles of conference proceedings, and the names and locations of publishers. It considers only those parts of the references that are not assigned to any data element yet. These unassigned parts mostly consist of the above data elements and the title of the referenced works.

Periodical name and publisher have very similar morphological properties and most references include either the name and location of a publisher or the name of a periodical. Note that we have not observed one case where both co-occur in one reference, and thus our algorithm does not consider this

Table 4 References with embedded references to volumes

Reference to	Example (embedded references in bold)
Book chapter	Thor, A. U. 2011: The title of the chapter. In: **Itor, E. D. (Ed.): The book title. Publisher, Place**: 8–15
Proceedings paper	Thor, A. U. 2011: The paper title. In:**Newton, G. et al. (Eds.): Proceedings of VLDB 2011, Seattle, WA, USA**

case. Embedded references to proceedings given completely with editors and/or publisher (see Table 3 for an example) are extracted in Step 4, this current step does not need to consider them. Standalone proceedings titles in turn exhibit the same morphological properties as names of periodicals. Thus, we do not need to have separate steps to extract names of periodicals, the titles of conference proceedings, and the names and locations of publishers, respectively. This extraction takes place in one step.

In general, all instances of the data elements sought in this step are in title case (i.e., all words start with a capital letter, usually except for stop words like determiners and prepositions), and RefParse uses patterns exploiting this morphological property to identify them. Furthermore, we have not observed any co-occurrences of abbreviated title case words and stop words, so our algorithm ignores this case. Names of periodicals are either written out in full, including intermediate stop words like prepositions and determiners, or they are abbreviated, omitting the stop words. RefParse exploits this to find the boundaries of potential abbreviated periodical names: namely, it cuts off abbreviated title case blocks at tokens preceded or followed by stop words. This happens for each reference in isolation, not for all references together. The reason is that the use of abbreviations is often inconsistent throughout a reference list. The decision rests on specific considerations like how much space is left in a line. So this step cannot rely on redundancy to determine the style of the data elements sought. The following example illustrates how RefParse finds the boundaries of abbreviated title case blocks:

> Consider the following reference string, comprising the (fictional) periodical name *Exploration and Discovery of Tropical Faunas* in abbreviated form. Steps 1–3 have already identified the author list and the base elements:
>
> *Thor, A. U. (1921) Results of the British expedition to the Amazonas region of Brazil. Expl. Disc Trop. Faun. 15: 8–55.*
>
> The part of this reference that has not been assigned to any data elements yet is *Results of the British expedition to the Amazonas region of Brazil. Expl. Disc Trop. Faun*. From the presence of the part designator 15, RefParse concludes that the referenced work is an article published in a periodical, and therefore seeks the name of the latter in the above string. It finds the abbreviated title case block *Expl. Disc Trop. Faun.* Morphologically, *Brazil* could be part of this title case block as well. However, it is preceded by the stop word of without any intermediate punctuation, and thus RefParse concludes that *Brazil* is not part of the abbreviated periodical name.

It can happen that a reference includes more than one title case block, i.e., a sequence of words in title case. RefParse handles such cases using the following heuristic. If a reference contains an abbreviated and a non-abbreviated title case block, the former becomes the periodical name or publisher. We have observed that titles of referenced works are extremely rarely abbreviated in reality, whereas abbreviations are relatively common for periodical names. If there are two or more unabbreviated title case blocks, the one closest to a given part designator becomes the periodical name. Here, the underlying observation is that in real-world references to journal articles, the part designators usually follow immediately after the journal name. If no part designator is present either, the rightmost title case block becomes the publisher name as a last resort, based on the observation that the publisher usually is closer to the end of a reference, in particular after the title.

> In Examples 1 and 1a, RefParse finds the title case blocks *The Particular Journal* and *Another Journal* as candidate periodical names, respectively. As both are the only title case blocks that have remained unassigned thus far (author names have been identified as such in Steps 2 and 3), the choice is clear.

In addition to the patterns and the selection heuristics based on the observations described in the previous paragraph, RefParse can use a dictionary of periodical and publisher names in this step. In addition, a statistics on how often individual words occur as part of such names and how often they occur elsewhere can help select the actual periodical or publisher name from several title case blocks. The dictionary facilitates recognizing title case blocks known to be names of periodicals or publishers from prior training or online learning. The statistics in turn model human intuition. In particular, this helps RefParse to recognize periodical names based on the presence of words frequently occurring in them. For instance, it would suggest that *Journal of the Science Museum of Somewhere*, even though unseen as a whole so far, is very likely a periodical name because it starts with *Journal*, just as many other periodical names do. While increasing accuracy, however, these additional sources of knowledge are not required for RefParse to work.

Step 6: Title extraction This step finally identifies the title of the referenced works. While the title arguably is the most important data element in bibliographic references, it is also the one whose instances exhibit the least regularity. In many reference styles, contemporary ones in particular, titles are not even in title case. This irregularity is the reason to identify them last, namely as the longest continuous part of a reference that has remained unassigned in all previous steps.

In Examples 1 and 1a, the remainders that are selected as the titles of the referenced works are *This is the article title* and *Some other article title*, respectively.

There is a notable exception to the length-based rule, however, namely with references to online resources. RefParse does not consider unassigned parts as the publication title if they mainly consist of dates. The following example illustrates the rationale behind this restriction:

In references to web resources, the plainly length-based heuristic can incur errors, namely in case of a very short title. Consider the following reference to a Wikipedia article:

Bibliography. Wikipedia, the Free Encyclopedia http://en.wikipedia.org/wiki/Bibliography *(accessed April 1st, 2012).*

The URL has been identified in Step 1, the publisher name *Wikipedia, the Free Encyclopedia* in Step 5. Now, plainly choosing the longest part that has remained unassigned thus far as the title would result in selecting *accessed April 1st, 2012* over *Bibliography* and thus incur an error. The date-based filter ensures choosing the actual title, namely *Bibliography*.

Step 7: User feedback In interactive mode, RefParse prompts users to double check parsing results and correct possible errors. This increases data quality in real-world deployments. Figure 5 shows such a dialog; each individual detail attribute RefParse extracts is encoded in a different color for easier distinction. Users can correct the parse by selecting pieces of the reference string and right clicking. The context menu then offers the actions available for the current selection. The explanation across the top of the dialog is intended to render using the dialog as simple as possible.

Step 8: Learning After users have double checked parsing results, RefParse can assume that there are very few remaining errors, if any. Thus, it is possible to use the parsing results as training examples to extend the dictionaries of person names, publisher names and locations, periodical names, and proceedings titles.

Remembering instances of other data elements for reuse in future parsing is unlikely to help: titles are relatively unique and thus are unlikely to reoccur. The possible instances of numeric data elements cover the entire language generated by their underlying alphabet of digits. Thus, morphological clues are sufficient to identify them reliably.

4.4 The infix match problem

In a preliminary evaluation we have observed an issue with dictionaries of periodicals and publisher names. This issue is counterintuitive at first glance: using dictionaries can actually be harmful to the parsing result. The underlying problem of this phenomenon is as follows: matches of dictionary lookups can prevent the complete periodical or publisher name from being found. It occurs in references comprising lengthy and relatively rare periodical names that have a shorter and more common periodical name as an infix. We call the phenomenon the *infix match problem* (IMP for short), illustrated in the following example:

Suppose that the dictionary of known periodical names contains the string *Nature*. This is actually quite likely, due to the importance of that periodical. Further, consider a reference to an article published in a volume of the *North Dakota Journal on the History and Development of Nature*. While

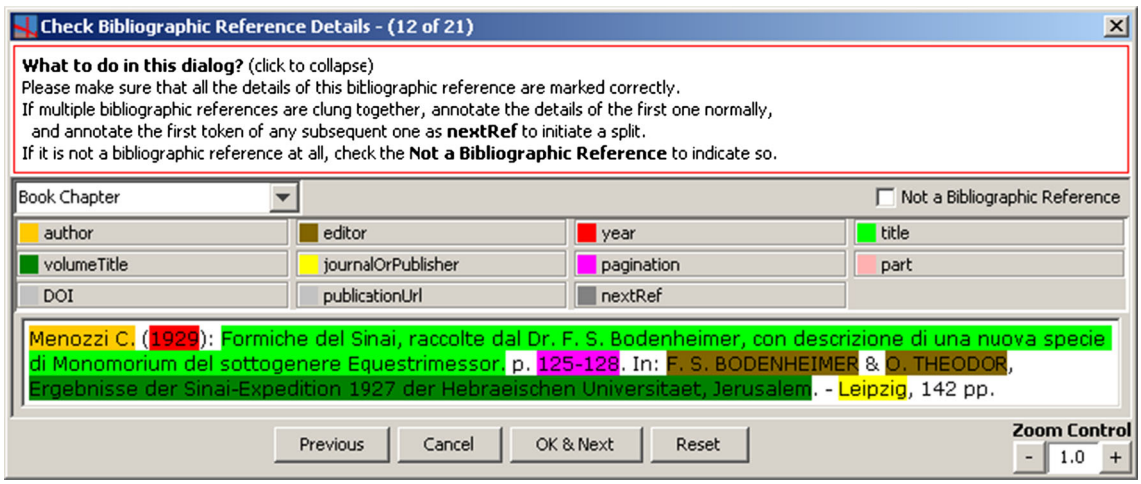

Fig. 5 User feedback dialog

this journal does not exist in actuality to our best knowledge, it perfectly suited to illustrate the IMP: a dictionary lookup for *Nature* results in a match. Using the known value *Nature* as the periodical name for that reference right away only captures a small part of the actual periodical name.

There is a simple heuristic that can mitigate the IMP: in most cases, the periodical name is surrounded by punctuation marks or numbers. In particular, it is often preceded by the terminal punctuation mark of the title, and it often has the part designators following right after it. In such cases, discarding dictionary lookup matches that lie in the middle of an uninterrupted sequence of words prevents the IMP from causing errors.

However, this heuristic does not work for references that give the periodical name in an abbreviated form, which usually involves numerous dots. Suppose the periodical name *North Dakota Journal on the History and Development of Nature* is abbreviated as *N. Dak. Jour. Hist. Dev. Nature*. In this case, the dictionary lookup match *Nature* is preceded by the dot after *Dev* and likely followed by part designators.

The IMP is not specific to RefParse; it rather affects all dictionary, statistics, and knowledge-based approaches to bibliographic reference parsing. In particular, it affects approaches that apply punctuation-based blocking techniques, which are easily misled by the dots in abbreviated periodical names.

To counter the IMP, RefParse uses an improved form of blocking; namely, it extracts unabbreviated blocks akin to existing approaches, but uses dedicated patterns to extract abbreviated blocks as a whole. It then does dictionary lookups only for entire blocks and for concatenations of successive blocks. The latter is to capture names of periodicals and publishers that include punctuation marks, e.g., *Springer, Berlin, Heidelberg*. In addition, RefParse uses a dedicated heuristic for the titles of proceedings volumes: the said titles almost always start with the word *Proceedings* or its abbreviation *Proc*. and have the conference year close to the end. RefParse thus generates a respective pattern, including the year of publication identified in Steps 1 and 3. If this pattern matches a part of a reference string, RefParse uses this part as an atomic block, regardless of internal punctuation.

Based on our observations of real-world data, RefParse then applies the following heuristics to assign the given blocks to the title and the periodical name or publisher:

- The first block belongs to the title, which almost always precedes the periodical name or publisher.
- If the title of a proceeding volume is present, it becomes the periodical name, together with all subsequent blocks.
- The titles of articles published in periodicals rarely include inner dots, only one at the end in several reference styles. In consequence, any further dotted parts likely belong to the abbreviation of a periodical name. RefParse exploits this for reference strings, Steps 1 and 3 have identified a part designator. This is because the latter is a good indication that the referenced work is an article published in a periodical. If a dictionary lookup match is embedded in such a likely abbreviated periodical name, RefParse ignores it to prevent the known infix match from incurring an error.
- On the other hand, names and locations of publishers are very rarely infixes of others, so the IMP rarely affects references to books and book chapters. Consequentially, if Steps 1 and 3 did not identify a part designator, and thus RefParse can rule out the presence of a periodical name, it does use dictionary lookup matches.

5 Evaluation

In this section, we report on a thorough evaluation of the RefParse algorithm. We use two test sets for our experiments, a contemporary one to facilitate comparisons with related approaches, and a historical one to challenge RefParse with the diversity found in this kind of real-world data.

5.1 Test data sets and metrics

The *Cora Corpus* [15] is our contemporary test data set; it has formerly been used in the evaluation of related work [10]. This corpus consists of 500 individual parsed references that are not organized in reference lists. We have prepared the references as described in [10], in particular breaking the author field down to individual authors and moving separators between the fields.

The *Plazi Corpus* is our test data set, stemming from a real-world document collection; it consists of nearly 25,000 bibliographic references extracted from the over 1,000 biological documents hosted by Plazi.[9] These documents have been published over the course of the last 250 years. They are written in five different languages (English, French, German, Portuguese, and Italian). The works they reference are mostly journal articles, proceedings papers, book chapters, and entire books. Due to the long time span and the many different origins, the reference styles used vary widely and cover many of the idiosyncracies of historical bibliographic referencing.

We assess the accuracy of RefParse at several levels of granularity, using the same metrics as [13]:

- The *word accuracy* measures the fraction of the words (including numbers and punctuation marks) of a bibliographic reference string that are assigned to the correct data element.

[9] http://plazi.org/.

Table 5 Results with the Cora Corpus (standard deviation in brackets where relevant)

	RefParse-g (%)	RefParse-d	ParsCit (%)	FreeCite (%)
Word/token	91.5	89.8 % (0.6 %)	83.0	83.8
Field				
Author	98.6	98.6 % (0.7 %)	95.7	95.7
Editor	74.6	78.6 % (6.3 %)	0	0
Title	79.0	74.5 % (1.3 %)	91.0	91.0
Year of publication	98.8	99.1 % (0.5 %)	96.7	96.7
Pagination	97.7	97.0 % (0.7 %)	88.9	1.6
Part dsignators	96.0	89.2 % (2.3 %)	66.7	96.0
Volume ttle	38.8	38.6 % (3.8 %)	46.3	50
Journal/publisher	68.0	61.6 % (1.8 %)	53.1	54.2
Instance	58.4	52.1 % (2.3 %)	23.4	12.2

– The *field accuracy* reflects how many of the data elements of a bibliographic reference string are identified correctly.
– The *instance accuracy* is the fraction of the bibliographic references for which all data elements are identified correctly.

The implementation of RefParse used in our experiments is available from https://git.scratchpads.eu/v/refparse.git under GPLv3, complete with configuration files containing all the patterns, etc.

5.2 Experimental setup

For each of the two test data sets, we have run four experiments:

The first experiment is in fully automated mode, i.e., without user corrections and subsequent learning, and thus with the dictionaries remaining empty throughout the experiment. This facilitates comparison to pattern-based parsers that do not make any use of learning. The results are labeled "RefParse-g", "g" for "generic".

The second experiment is with domain knowledge, i.e., with dictionaries of person names, publishers, and periodicals, as well as word-level statistics for the latter. We have randomized document order, split the data sets in half, prefilled the dictionaries with the data element instances from the first half, and then run the actual evaluation on the second half. This facilitates comparison to learning-based parsers. Because the division of the data sets influences the learning and thus the results, we have run this experiment five times with five different randomized document orders and have averaged the results. This setup is similar to a fivefold cross-validation [5], but in fact more challenging. This is because a standard cross-validation would use 80 % of the data for training and only 20 % for test, whereas we divide the data equally between training and test, thus using less training data. The results are labeled "RefParse-d", "d" for "dictionaries".

The third experiment is in interactive mode, i.e., with user interaction and thus with online learning. We have simulated user corrections by providing RefParse with the gold standard document after measuring the accuracy of its output, and before proceeding to the next document. As in the previous experiment, the order of the documents determines when RefParse learns which data element values and thus influences results. We have thus run this experiment on the same five randomized document orderings used in the second experiment and have averaged the results. The results are labeled "RefParse-i", "i" for "interactive".

While the discussions of the first three experiments already feature a comparison with figures reported in a related work, the fourth experiment compares RefParse to the state of the art by means of actual evaluation runs of competing approaches on our two test corpora. In particular, we have run our test data through a ParsCit [7] web service and a FreeCite web service and have measured their result accuracy. The results are labeled "ParsCit" and "FreeCite", respectively. They reflect the state of the art as currently available online.

5.3 Experimental results

Experiments with Cora Corpus (Table 5) In this experiment, RefParse has to work with individual reference strings and thus has no repeated patterns to exploit. This causes it to degenerate into a purely pattern-based parser. Despite this disadvantage, RefParse clearly outperforms both ParsCit and FreeCite, even though the latter two were trained on the Cora Corpus according to the descriptions on the web pages. Only for the title and volume title, ParsCit and FreeCite are more accurate than RefParse. The results for RefParse also are in the ranges of the accuracy reported for other data sets in related work. Note that the latter mostly measure parsing quality as precision, recall, and f-score. However, these fig-

Table 6 Results with the Plazi Corpus (standard deviation in brackets where relevant)

	RefParse-g (%)	RefParse-d	RefParse-i	ParsCit (%)	FreeCite (%)
Word/token	94.3	96.6 % (0.1 %)	96.6 % (0.1 %)	78.9	79.7
Field					
Author	97.2	97.8 % (0.3 %)	97.8 % (0.3 %)	88.3	88.0
Editor	83.7	85.2 % (5.2 %)	88.6 % (6.8 %)	0	0
Title	78.4	90.7 % (0.4 %)	90.1 % (0.4 %)	40.4	32.4
Year of publication	99.5	99.7 % (0.06 %)	99.6 % (0.09 %)	95.5	89.7
Pagination	99.3	99.3 % (0.09 %)	99.3 % (0.09 %)	20.4	0.3
Part designators	97.7	95.3 % (0.2 %)	95.3 % (0.2 %)	42.0	64.3
Volume title	63.2	73.1 % (3.7 %)	74.1 % (3.8 %)	0.6	0.3
Journal/publisher	76.6	88.3 % (1.0 %)	88.3 % (1.0 %)	54.3	44.3
Instance	69.9	82.0 % (0.8 %)	81.8 % (0.8 %)	65.6	3.4

ures are easily converted to accuracy, which we have done to facilitate comparison. Namely, accuracy is equal to the product of precision and recall, whereas f-score is their harmonic mean:

$$\text{accuracy} := \text{precision} \times \text{recall}$$

Experiments with Plazi Corpus (Table 6) Even without dictionaries, i.e., solely relying on the structure inference and morphological clues, RefParse clearly outperforms all previous approaches. The accuracy for the title and the journal name are considerably better than reported in [1] for the plainly pattern-based ParaCite. We have to rely on the figures reported in [1] because we have not found an operational ParaCite service or an installation package. The improvement in accuracy clearly emphasizes the benefit of structure inference. With dictionaries, the accuracy increases for the respective fields and for other fields as well. In interactive mode, the results are similar to the experiments with training, even though the dictionaries are empty at the start of the respective experiments and are populated only over time.

Significance of results The significance of experimental results may suffer from statistical effects whenever experiments involve randomization. In particular, this affects the second and third experiment due to the random partitioning and the randomized document ordering, respectively. To assess the significance of their results, we have computed the standard deviation of our measurements. Tables 5 and 6 give it in brackets next to the averaged results, namely the "RefParse-d" and "RefParse-i" columns. For the experiments with the Plazi Corpus in particular, the standard deviation is very small for word and instance accuracy, as well as for most of the individual fields. This corroborates the significance and validity of our results. Only the editor and volume title fields exhibit a somewhat higher standard deviation. We hypothesize that this is because these two fields exist in relatively few references. So, randomization might actually have an impact there.

5.4 Performance of individual steps

Before discussing the evaluation results in general, we address the individual steps RefParse consists of. In particular, we use the accuracies achieved in the individual fields as indicators for the performance of the steps extracting the respective elements.

- Steps 1 and 2. The accuracies RefParse achieves for the base elements on the Plazi Corpus corroborate the validity of the approach to start with these elements. Especially, the accuracies for the morphologically highly distinctive elements year of publication and pagination are extremely high, exceeding 99 % in each experiment. But part designators and author names are extracted highly accurately as well, always exceeding 95 and 97 % of accuracy, respectively.
- Step 3. These results in combination also show the power of reference style identification by means of mutual alignment of possible data element arrangements. In particular, if the latter would not be able to reliably identify the reference style in use, the numerical elements would be disambiguated erroneously, incurring far less accurate results.
- Step 4. The accuracy achieved for the editor and volume title fields is a good indicator for the performance of host reference extraction. There are equal improvements regarding the editor field between the generic and the training-based and the training-based and the online learning modes, around 3 % each. This indicates the utility of dictionaries for this field. The improvement in the volume title field is a lot higher between the generic and the training-based or learning modes. However, we believe that this is mostly due to a gain in accuracy in

Step 5, for the same reasons as in the title field. See Step 6 below for an explanation.
- Step 5. The increase in accuracy in the periodical name/publisher field between the generic mode (76.6 %) and the modes that involve training or online learning (88.3 % in either) indicates as well that using dictionaries and word-level statistics for this data element is advantageous.
- Step 6. This also reflects in the accuracies RefParse achieves for the title and volume title fields, with equally high increases. Considering the fact that title/volume title and periodical name/publisher are adjacent in many reference styles, this behavior is intuitive; namely, an erroneous split between these fields incurs errors in both, and a correct one results in either field being extracted correctly.
- Steps 7 and 8. Do not extract any data elements, but rather obtain corrections from users, and extend the dictionaries from the human-approved parse results. Thus, there are no performance figures to discuss.

Overall, the accuracy regarding the individual data elements proves the design of RefParse to be very well suited to the problem. Namely, this is to move from easy to identify data elements to harder ones while exploiting repetitions and redundancies between individual reference strings wherever possible. In addition, the experimental results regarding the individual modes demonstrate that, as expected, the steps involving dictionaries and statistics benefit considerably from training and online learning. This highlights the suitability of the RefParse algorithm, especially in the interactive mode.

5.5 Discussion

With or without dictionaries, with or without training or user interaction and incremental online learning, RefParse significantly outperforms existing approaches. This generally applies to both token and instance accuracy. For specific data elements, namely title and volume title, either ParsCit or FreeCite outperforms RefParse on the Cora Corpus. However, bear in mind that the Cora Corpus consists of unrelated individual references. This causes RefParse to degenerate into a mere pattern-based parser.

We speculate that the reason for the better performance of ParsCit and FreeCite on the title and volume title fields lies in the way the individual algorithms learn: ParsCit and FreeCite use statistical models that indicate how likely a given token belongs to the title, volume title, or any other field. RefParse, on the other hand, only uses a statistic that indicates how likely a given token belongs to the name of the journal or publisher, and how likely this is not the case; it regards the title and volume title fields as absolutely free text. Without repeated patterns to exploit for identifying element boundaries, this very open approach can incur errors.

On the Plazi Corpus, where RefParse gets to work on entire reference lists and thus finds repeated patterns to exploit, it performs considerably better than both ParsCit and FreeCite for each individual field.

The heuristics devised in Sect. 4.4 counter the infix match problem effectively. The accuracy for the title, volume title, and periodical name/publisher fields increases significantly, both with training and with online learning. Namely, the improved blocking scheme prevents dictionary lookups for infixes of periodical names altogether. If none of the blocks occur in a dictionary, in turn, the statistics indicating how likely individual words are part of a periodical name help determine which of the blocks is most likely to actually be the periodical name.

Generally problematic are reference lists from low-quality OCR, in particular lists whose punctuation is recognized incorrectly or not at all. However, these cases pose extreme challenges for blocking based parsers as well, so this problem is not specific to RefParse. Problematic for RefParse in particular are reference lists that deviate from their style guide, mostly due to sloppy editing. Other approaches are less sensitive to this case: They do not exploit repeated patterns, and thus they do not depend on reference lists to be formatted uniformly in one style.

The real strength of RefParse shows in the interactive mode: without any initial training, the result accuracy increases quickly to very high levels. One might argue that accuracy is less important if users double check and correct the parsing results anyway. However, according to [21], the accuracy given before user interaction has a strong influence on the accuracy of the final result, i.e., after user interaction.

From another angle, however, this experiment also shows that learning from user corrections—a form of online learning—completely alleviates the need for labeled training data. Because the latter is the major bottleneck of training-based approaches and this constitutes a considerable advantage. In addition, it results in a continuous improvement and refinement of the statistics, which cannot happen with pre-deployment learning. The practice to learn online while processing data in a productive environment, as opposed to pure pre-deployment learning, is relatively popular in time series prediction [17], e.g., for stock prices, for the above reasons. RefParse demonstrates that this approach is also promising for bibliographic reference parsing, even though the latter does not involve the sequence aspect of the former.

Another recent development renders the online learning approach implemented in RefParse yet more interesting: community-based data collection and curation efforts have become more and more popular in recent years [20]. Using

this approach to compile online bibliographies requires community members to contribute, i.e., upload bibliographic references and to parse the ones contributed by others. In such a scenario, the more accurate the parsing result presented to users for checking, the less actual user intervention is required to achieve a desired accuracy in the final result. Because recognizing a correct parse takes less time than correcting an erroneous one, this reduces user effort. Due to this relationship, the strong performance of RefParse in the interactive mode makes it a very promising choice for community-based efforts aimed at bibliography compilation.

6 Conclusions

The extraction of data elements like author names and titles from bibliographic references of scientific publications is an essential task in the compilation of comprehensive bibliographies. Parsing bibliographic reference strings into their individual data elements also is a core prerequisite for advanced processing of bibliographic data, like entity matching [12] and duplicate reconciliation [14], online integration of data from different sources [4,16], author identification [9], and bibliometric analyses [6]. Furthermore, it facilitates the reuse of bibliographic records as metadata for digital objects stored in the respective repositories, in particular digital libraries and digital preservation efforts.

Most existing techniques handle references individually, one by one, and thus disregard the important additional information that lies in the fact that all references in a bibliography of a given work follow the same style. As a consequence, they are only successful with the highly standardized reference styles used in the last two decades; they cannot adapt to the wide variety of reference styles used in older publications.

To reliably extract detailed data from both historical and contemporary bibliographic reference lists, this paper has introduced RefParse, a novel parsing algorithm. RefParse is engineered based both on a thorough investigation of real-world data and on existing approaches to bibliographic reference parsing. In addition, RefParse has several distinctive new properties: (1) it handles entire lists of references together to exploit the structural similarities that exist between individual reference strings, (2) it infers the reference style at hand at runtime, which renders it independent of pre-acquired knowledge and any particular reference style, and (3) it can tightly integrate data quality control in the parsing process by means of user interaction and learn from corrections.

The blocking-based heuristics we have devised to overcome the infix match problem work very well: with their help, the use of training/online learning and dictionaries has the intended effect, considerably increasing accuracy for the titles, volume titles, and names of periodicals and publishers.

Our evaluation has shown that RefParse by far outperforms existing approaches with both contemporary and historic reference lists. The online learning in interactive mode yields considerable gains in accuracy, increasing instance accuracy to 82 %. This is particularly helpful in large-scale real-world applications, like the Biodiversity Heritage Library [8] or RefBank [23].

Despite the high accuracy RefParse already achieves, we also see room for further improvements. Namely, so far RefParse only works on plain text reference strings to achieve the maximum generality possible. It thus ignores clues included in reference strings that come in richer data formats like RTF or HTML and could help to further increase accuracy. In particular, reference strings that come in one of these richer formats may include changes in font style or even font face between different data elements, which might be used as additional delimiters.

Acknowledgments This research has received funding from the Seventh Framework Programme of the European Union (FP7/2007-2013) under Grant Agreement No. 261532 (ViBRANT—Virtual Biodiversity Research and Access Network for Taxonomy). The RefParse algorithm is in productive use as part of RefBank (RefBank: http://vbrant.ipd.uka.de/RefBank/), the platform of ViBRANT to build its Bibliography of Life.

References

1. Chen, C.-C., Yang, K.-H., Kao, H.-Y., Ho, J.-M.: BibPro: a citation parser based on sequence alignment techniques. In: Proceedings of AINAW, pp. 1175–1180, Okinawa (2008)
2. Cortez, E., da Silva, A.S., Goncalves, M.A., Mesquita, F., de Moura, E.S.: Fluxcim: flexible unsupervised extraction of citation metadata. In: Proceedings of JCDL, pp. 215–224, Vancouver (2007)
3. Day, M.-Y., Tsai, R.T.-H., Sung, C.-L., Hsieh, C.-C., Lee, C.-W., Wu, S.-H., Wu, K.-P., Ong, C.-S., Hsu, W.-L.: Reference metadata extraction using a hierarchical knowledge representation framework. Decis. Support Syst. **43**, 152–167 (2007)
4. Fuhr, N., Klas, C.P., Schaefer, A., Mutschke, P.: Daffodil: an integrated desktop for supporting high-level search activities in federated digital libraries. In: Proceedings of ECDL, pp. 597–612, Rome (2002)
5. Geisser, S.: Predictive Inference. Chapman and Hall, New York (1993)
6. Giles, C.L., Bollacker, K.D., Lawrence, S.: CiteSeer: an automatic citation indexing system. In: Proceedings of DL, pp. 89–98, Pittsburgh (1998)
7. Giles, C.L., Councill, I., Kan, M.-Y.: ParsCit: an open-source CRF reference string parsing package. In: Proceedings of LREC, Marrakech (2008)
8. Gwinn, N.E., Rinaldo, C.: The Biodiversity Heritage Library: sharing biodiversity literature with the world. IFLA J **35**(1), 25–34 (2009)
9. Han, H., Zha, H., Giles, C.L.: Name disambiguation in author citations using a k-way spectral clustering method. In: Proceedings of JCDL, pp. 334–343, Denver (2005)
10. Hetzner, E.: A simple method for citation metadata extraction using hidden markov models. In: Proceedings of JCDL, pp. 280–284, Pittsburgh (2008)

11. Huang, I.-A., Ho, J.-M., Kao, H.-Y., Lin, W.-C.: Extracting citation metadata from online publication lists using BLAST. In: Proceedings of PAKDD, pp. 539–548, Sydney (2004)
12. Köpcke, H., Rahm, E.: Frameworks for entity matching: a comparison. Data Knowl Eng **69**(2), 197–210 (2010). doi:10.1016/j.datak.2009.10.003
13. Krämer, M., Kaprykowsky, H., Keysers, D., Breuel, T.: Bibliographic meta-data extraction using probabilistic finite state transducers. In: Proceedings of ICDAR, pp. 609–613, Curitiba (2007)
14. Manguinhas, H., Borbinha, J.: Quality control of metadata: a case with UNIMARC. In: Proceedings of ECDL, pp. 244–255. Alicante (2006)
15. McCallum, A., Nigam, K., Rennie, J., Seymore, K.: A machine learning approach to building domain-specific search engines. In: Proceedings of IJCAI, pp. 662–667, Stockholm (1999)
16. Mutschke, P.: Enhancing information retrieval in federated bibliographic data sources using author network based stratagems. In: Proceedings of ECDL, pp. 287–299, Darmstadt (2001)
17. Palit, A.K., Popovic, D.: Computational Intelligence in Time Series Forecasting: Theory and Engineering Applications (Advances in Industrial Control). Springer, New York (2005). ISBN 978-1-84628-184-6
18. ParaCite. http://paracite.eprints.org/
19. Patashnik, O.: BibTeXing—the original manual. In: Proceedings of the IEEE, vol. 77 (1988)
20. Rosner, H.: Data on wings. Sci. Am. **308**(2), 68–73 (2013)
21. Sautter, G., Böhm, K.: High-throughput crowdsourcing mechanisms for complex tasks. In: Proceedings of SocInfo, LNCS. vol. 6984, pp. 240–254, Singapore (2011)
22. Sautter, G., Böhm, K.: Improved bibliographic reference parsing based on repeated patterns. In: Proceedings of TPDL, pp. 370-382. Paphos (2012)
23. Sautter, G., King, D., Morse, D.: Towards a universal bibliography—the RefBank approach. In: Proceedings of TDWG, Beijing (2012)
24. Takasu, A.: Bibliographic attribute extraction from erroneous references based on a statistical model. In: Proceedings of JCDL, pp. 49–60, Houston (2003)
25. ViBRANT: Virtual Biodiversity Research and Access Network for Taxonomy, grant 261532 in EU FP7/2007–2013
26. Yeates, S., Witten, I.H., Bainbridge, D.: Tag insertion complexity. In: Proceedings of Data Compression Conference, pp. 243–252, Snowbird, Utah (2001)