Extraction Algorithms for Hierarchical Header Structures from Spreadsheets

Keisuke Goto
Fujitsu Laboratories Ltd.
Kawasaki, Japan
goto.keisuke@jp.fujitsu.com

Yuiko Ohta
Fujitsu Laboratories Ltd.
Kawasaki, Japan
yuiko@jp.fujitsu.com

Hiroya Inakoshi
Fujitsu Laboratories Ltd.
Kawasaki, Japan
inakoshi.hiroya@jp.fujitsu.com

Nobuhiro Yugami
Fujitsu Laboratories Ltd.
Kawasaki, Japan
yugami@jp.fujitsu.com

ABSTRACT

Spreadsheets are widely used to manage statistical data, and a massive amount of valuable spreadsheets are now public as open data. Spreadsheet headers may contain hierarchical structures, and values in spreadsheets are associated with not only headers at the same column or row but also those corresponding to their ancestors. When integrating spreadsheets with other datasets, it is necessary to extract hierarchical header structures and associate this information with values.

In this paper, we propose three algorithms to extract hierarchical structures from spreadsheets based on the relation of values. Since our algorithms are focused only on values, they can be applied to heterogeneous spreadsheets such that the structures of row or column headers are not explicitly indicated by specific header features such as space, indent, or keywords. Our experiments showed that the F-measure for extractions with one of our algorithms is 0.73. The runtimes of our algorithms are practical for most datasets; therefore, they can process 995 real spreadsheets of up to 10^5 rows, which accounts for 99.5% of 1000 spreadsheets randomly chosen from PublicData.eu, within 16200 seconds (the average runtime per file is 16.28 seconds).

1. INTRODUCTION

Governments and organizations are opening up their own statistical data so that anyone can analyze them and create new applications. Many portal sites have been launched [1], and they are distributing enormous amount of data in various formats such as CSV, Excel, PDF, HTML, and RDF. The Semantic Web and Linked Data communities are advocating that open data be published as RDF format as default since it is suitable for integrating several datasets, and enables the querying of any values by using the uniform query language SPARQL [1, 6]. However, most open statistical data are published in spreadsheet format such as CSV or Excel, which are not suitable for integrating and querying (see Table 1). To promote the use massive of large amounts of statistical data, it has to be transformed to RDF format.

Many tools for transforming spreadsheets into RDF have been published [2] . A spreadsheet consists of three parts, row headers, column(col) headers, and values, and a value at row i and col j associated with the row header at i and the col header at j. Most tools automatically transform a spreadsheet into RDF by associating each value with the row and col headers at the same row and col of that value. However associated information on the above transformation is insufficient for representing the value rigorously when the row or col headers have hierarchical structures. A hierarchical structure of headers in a spreadsheet shows that a concept indicated by the header corresponding to the parent is divided into more specific concepts indicated by headers corresponding to the children. In this case, the value at row i and col j associated with not only the row header at i and the col header at j but also the headers corresponding to their ancestors. Therefore the above transformation may not include sufficient information of values when headers have hierarchical structures.

Chen and Cafarella [3] proposed a method to extract hierarchical structure of spreadsheets. They considers the extraction problem as binary labeling problem to set labels true or false to all pairs of parent and child headers. Their method uses features of each header, such as fonts, spaces, indents, italic, bold, or capitalization, and extract structures by Support Vector Machine [4]. This approach is effective for spreadsheets whose structures are specifically indicated by the above features written by owners but not for those which are not indicated by such features.

In contrast to most approaches concentrating on the features of header information, we focus on the features of values. A hierarchical structure of headers shows that a concept indicated by a parent header involves being divided into more specific concepts indicated by other child headers. An

[1] 520 open data portals are now listed in `http://dataportals.org`

[2] Over 30 tools that transform spreadsheets into RDF are listed in `https://github.com/timrdf/csv2rdf4lod-automation/wiki/Alternative-Tabular-to-RDF-converters`.

Portal site	CSV	Excel	PDF	HTML	RDF
http://data.gov.uk	3675	1697	665	523	211
http://catalog.data.gov	11168	966	32640	70039	7071
http://publicdata.eu	10868	6211	1527	2335	2032
http://data.go.jp	656	3086	7926	5961	2
http://open.canada.ca	7004	1191	149182	5587	1

Table 1: Five portal sites distributing open data in several formats. The number of available files for each format is measured in December 2015.

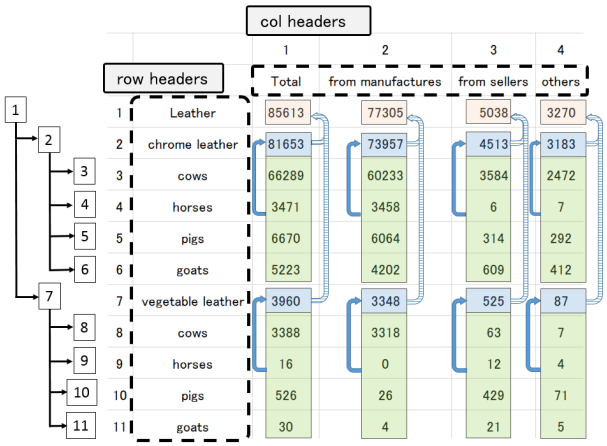

Figure 1: Spreadsheet showing number of leather items classified by their origins, and sellers selling them.

important observation is that for a structure, a value corresponding to the parent header is also divided into values corresponding to the child headers in many cases, and the parent value equals the sum of child values. For a row i and $j..k$, if the value corresponding to row i equals the sum of values corresponding to rows $j..k$ for each col, we call this a *summation relationship* satisfying i and $j..k$. Our approach extracts such relationships only by values then outputs the hierarchical structure of headers corresponding to the relationships. Since we only focus on the values, our approach can be applied to spreadsheets that do not have specific features for structures written by owners. In the rest of the paper, we only consider the hierarchical structures of row headers, but our approach can also be applied to those of cols by transposing input spreadsheets.

We propose three algorithms, naive, sample, and non-negative, and we evaluated them in terms of extraction accuracy and runtime performance by applying them to real spreadsheets from PublicData.eu [3]. Our experiments showed that F-measure for extractions with one of our algorithms is 0.73 and the runtimes of sample and non-negative algorithms are practical for most datasets; therefore, they can process 995 real spreadsheets of up to 10^5 rows, which accounts for 99.5% of 1000 spreadsheets randomly chosen from PublicData.eu, within 16200 seconds (the average runtime per file is 16.28 seconds).

2. RELATED WORK

The transformation of spreadsheets into more rigid formats such as RDF or relational data base has been mainly investigated in the database community for the objective to facilitate the integration. There are three traditional approaches. A schema matching approach [2, 9] transforms spreadsheets into a database by specifying the attribute mapping. A rule based approach [7], transforms spreadsheets into a database according to user-provided rules. A user-interface approach [8, 10] transforms spreadsheets into a database by user interactive operation using interactive user interfaces. The drawback of the above approaches is that human effort is required to specify rules or operate transformation. Recently a crowd-sourcing approach which is suitable for massive spreadsheets was proposed [5]. This approach firstly creates a default mapping from spreadsheets to RDF using a existing mapping tool, and then users correct the mapping errors by using wiki system. Though this approach requires manual operations, it is able to scale for massive spreadsheets by using crowd-sourcing.

[3] http://publicdata.eu

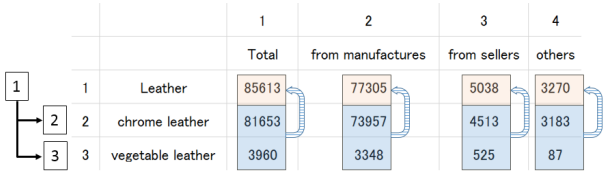

Figure 2: Sub spreadsheet of that in Figure 1 whose rows do not appear as child rows of the structures of depth 1, $(2, 3...6)$ and $(7, 8...11)$.

3. PRELIMINARIES

3.1 Notation

For a one-dimensional array $A[1..n]$ of length n, $A[i]$ indicates the i-th element of A, and $A[i..j] = A[i]A[i+1]..A[j]$ is the sub array of A between positions i and j inclusive. For a two-dimensional array $T[1..n][1..m]$ of n rows and m cols, $T[i][j]$ indicates the element of T at row i and col j, $row(T, i) = T[i][1]T[i][2]..T[i][m]$ indicates the one-dimensional array extracted from row i from T, $col(T, i) = T[1][i]T[2][i]..T[n][i]$ indicates the one-dimensional array extracted from col i from T. For ease of explanation, we call a one-dimensional array and a two-dimensional array just an array and a table, respectively.

3.2 Spreadsheets

A Spreadsheet of n rows and m cols consists of two string arrays, row headers of length n and col headers of length m, and a numerical table of n rows and m cols, indicated as $rheader[1..n]$, $cheader[1..m]$, $T[1..n][1..m]$, respectively. Each value $T[i][j]$ is associated with $rheader[i]$ and $cheader[j]$. In

Figure 1, the value '77305' at row 1 and col 2 is associated with 'leather' of the row header at row 1 and 'from manufactures' of the col header at col 2, and indicates the total number of leather items from manufactures.

Spreadsheets may contain hierarchical header structures, and in such case, the value at row i and col j associated with not only the row header at i and the col header at j but also the headers corresponding to their ancestors. For example, row headers of the spreadsheet in Figure 1 have a hierarchical structure of depth 2, 'Leather' at row 1 has two children 'chrome leather' at row 2 and 'vegetable leather' at row 7, and each also has children at rows 3..6 and 8..11 respectively. A structure of parent row i and the preceding or succeeding child rows $j..k$ is expressed as $(i, j...k)$. Let $T_1 = T$, $L_0 = \emptyset$, and for $x \geq 1$, let L_x be the set of structures of T_x and T_{x+1} be the sub table of T_x whose rows do not appear as child rows in L_x (see Figure 2). For a table T, its *hierarchical* structures are expressed as a list of sets of structures for each depth $L = L_1, L_2, ..., L_h$, where L_x is the set of structures of depth x. For example, the structures in Figure 1 are represented as $L = L_1, L_2$, where $L_1 = \{(2, 3...6), (7, 8...11)\}$, $L_2 = \{(1, 2...3)\}$ (the row indices $1, 2, 3$ of T_2 correspond to the row indices $1, 2, 7$ of T_1). The indices of L_x for $x > 1$ are different from the initial table T_1, but the indices corresponding to T_1 in L_x can be easily obtained by associating indices in L_x through T_{x-1} to T_1 recursively. Note that the hierarchical structures we consider in this paper are represented as trees, and the indices of each parent and its children are consecutive.

3.3 Summation Relationship

In this section, we give a more precise definition of summation relationship. For ease of presentation, we only treat the summation relationship when child rows succeed the parent row, but it is the same for when the child rows precede the parent row. For a table $T[1..n][1..m]$ and $1 \leq i < j \leq n$, row i and its succeeding rows $i + 1$ to j satisfy a *strict summation relationship* when $T[i][c] = \sum_{i<k\leq j} T[k][c]$ for all cols c. Unfortunately, a strict summation relationship does not suit our objective because real spreadsheets may contain errors such as rounding errors, measuring errors, or human errors, so $T[i][c]$ may be slightly different with $\sum_{i<k\leq j} T[k][c]$. There is also a case in which the summation relationship satisfies most but not all cols.

To avoid these problems, we adopt a more robust definition of *summation relationship* by using two criteria, error rate and satisfaction rate. For an array $A[1..n]$, $1 \leq i < j \leq n$, $erate_A(i, j)$ indicates the error rate of the parent value $A[i]$ and sum of child vales $A[i + 1..j]$.

$$erate_A(i, j) = 1 - \frac{\min(A[i], \sum_{i<k\leq j} A[k])}{\max(A[i], \sum_{i<k\leq j} A[k])}$$

This rate varies from 0 to 1, and $erate_A(i, j) = 0$ means that they exactly match. For a real number e, we say that the summation relationship with error rate e is satisfied for $A[i]$ and $A[i+1..j]$ if $erate_A(i, j) \leq e$, and the relation is denoted as $A[i] \approx_e A[i + 1..j]$. For a table T of n rows and m cols, $1 \leq i < j \leq n$, and a real number e, $srate_{T,e}(i, j)$ indicates the satisfaction rate of cols that satisfies the summation relationship between row i and rows $i + 1..j$.

$$srate_{T,e}(i, j) = \frac{|\{c|col(T, c)[i] \approx_e col(T, c)[i + 1..j]\}|}{m}$$

For real numbers e and s, we say that the summation relationship with error rate e and satisfaction rate s is satisfied for row i and rows $i + 1..j$ if $srate_{T,e}(i, j) \geq s$, and the relation is denoted as $T[i][1..m] \approx_{e,s} T[i + 1..j][1..m]$.

This new definition is more robust than that of the strict one. However this robust definition may result in the extraction of undesired structures. In such a case, some child rows overlap, and the desired structure may be buried among them. We discuss how to avoid the problem in Section 4.1

We further define the hierarchical summation relationship. For a table T, let $T_1 = T$, $L_0 = \emptyset$. The hierarchical summation relationship of T is the list of the sets of summation relationships for each depth $L = L_1, L_2, ..., L_h$, where L_x for $x > 0$ is a set of the summation relationships for T_x whose elements do not overlap, and T_{x+1} for $x > 0$ is the sub table of T_x whose rows do not appear as child rows in L_x.

4. OVERVIEW OF ALGORITHMS

Our objective is to compute the hierarchical summation relationship for a given table T and output them as hierarchical structures of T. We tackle the following problem and propose three algorithms to solve this problem.

PROBLEM 1. *Given a table $T[1..n][1..m]$, and real numbers e, s, compute the hierarchical summation relationship $L = L_1, L_2, ..., L_h$ with error rate e and satisfaction rate s for T.*

Since our algorithms have many common parts, we first give an overview of our algorithms then describe those common parts. The difference in the algorithms are discussed in Section 5.

An overview of the algorithms is given in Algorithm 1. Our algorithms recursively process to extract the hierarchical summation relationship. For each recursive step x, the algorithms take the following three steps. Step 1 at Line 4 computes the set of all summation relationships S_x for a spreadsheet T_x. Some child rows of S_x may overlap, so Step 2 at Line 5 computes the subset L_x of S_x whose elements have higher reliability than the deselected ones and whose child rows do not overlap. If L_x is empty, T_x does not contain rows that satisfy the summation relationship, then outputs the hierarchical summation relationship $L = L_1, ..., L_{x-1}$, and the algorithm terminates (See Line 6, 10). Otherwise, T_{x+1}, which are deleted child rows of L_x from T_x , may contain summation relationships, so Step 3 at Line 9 deletes child rows of L_x from T_x and recursively processes the new table T_{x+1}.

The difference between the three algorithms is only in Step 1 involving the extraction of the summation relationship S_x from a table T_x, which is the most bottlenecked part of the algorithm. Steps 1 and 2 are described in Sections 5, 4.1, respectively. Step 3 can be done by simply deleting child rows of L_x from T_x in $O(n)$ time.

For a spreadsheet of n rows and m cols, our three algorithms take the h step recursively, where h is the maximum depth of the output structures, and for each recursive step x, all of three take $O(|S_x|m + |S_x| \log |S_x|)$ for the common parts Step 2 and 3. Each algorithm additionally takes $O(n^2 m)$, $O(dn^2 + nm + |M_x|m)$, and $O(dn \log n + nm + |M_x|m)$ times, respectively in Step 1, where d is a parameter which is very small, and M_x is the candidate set of S_x whose size can be $O(n^2)$ in the worst case, but is practi-

Algorithm 1: Extracting the hierarchical summation relationship from a table

Input: Table $T[1..n][1..m]$, and real numbers e and s
Output: Hierarchical summation relationship of T

```
T_1 ← T ;
L ← empty list ;
for x ← 1 to n do
    S_x ← extract(T_x, e, s) // extracts summation
        relationship with error rate e and
        satisfaction rate s for T_x
    L_x ← filter(S_x, e, s) // computes subset of S_x
        whose child rows do not overlap
    if |L_x| = 0 then
        Break;
    L append L_x ;
    T_{x+1} ← delete_rows(T_x, L_x) // deletes rows from
        T_x that appear as child rows in L_x
Output L ;
```

cally small. The detail of algorithms and parameters are described in Section 5.

4.1 Filtering

For a set of summation relationships S_x, we give a comparison criterion for relationships to compute the subset L_x whose child rows do not overlap and are more reliable than the deselected ones. Intuitively, a relation satisfying a higher satisfaction rate looks more reliable than that satisfying a lower satisfaction rate, and there is the same expectation for lower and higher error rates for the relation of each col.

For two summation relationships, we say that a relation is more reliable than the other if (1) its satisfaction rate is higher or (2) the rate is the same and its average error rate is lower. The satisfaction and average error rate for all relationships of S_x can be computed in $O(|S_x|m)$ time, and all these relationships of S_x can be sorted in $O(|S_x| \log |S_x|)$ time. We sort S_x by the above criterion then greedily select relationships from S_x in order according to reliability, but not overlapping relationships that have been selected previously. Computation of L_x from S_x can be done in $O(|S_x|m + |S_x| \log |S_x|)$ time in total.

5. EXTRACTION ALGORITHMS

For a table T of n rows and m cols, our three algorithms extract the summation relationships of T. The naive algorithm takes into account all $O(n^2)$ combinations of rows and verify whether the combination satisfies the summation relationship for each of m cols. The sample algorithm reduces the amount of verifying rows by sampling d cols and runs faster than the naive algorithm, where the parameter d is set by users. The non-negative algorithm can be applied only for tables with only non-negative values, but it runs faster than the sample algorithm.

5.1 Naive Algorithm

For a table $T[1..n][1..m]$, the naive algorithm takes into account all $O(n^2)$ combinations of rows then validates whether each combination satisfies the summation relationship. Let C be a table of $n+1$ rows and m cols such that $C[1][k] = 0$ for $1 \leq k \leq m$ and $C[r][c] = \sum_{1 \leq k < r} T[k][c]$ is the accumulation of values from row 1 to row $r-1$ on col c for $1 < r \leq n+1$ and $1 \leq c \leq m$. If we have C, for a col c, row i, and rows $i+1..j$, $\sum_{i<k\leq j} T[k][c] = C[j+1][c] - C[i+1][c]$ can be computed in $O(1)$ time, and the summation relationship $T[i][1..m] \approx_{e,s} T[i+1..j][1..m]$ can be verified for row i and rows $i+1..j$ in $O(m)$ time. Since C can be computed in $O(nm)$ time, the naive algorithm can compute the set of summation relationships of T in $O(n^2 m)$ time.

5.2 Sample Algorithm

For a table $T[1..n][1..m]$ of n rows and m cols, an important observation is that when $T[i][1..m] \approx_{e,s} T[i+1..j][1..m]$ is satisfied on T, the summation relationship is satisfied in most cols. The sample algorithm randomly samples d cols from T and naively extracts the summation relationships $(i, i+1...j)$ satisfying $col(T,c)[i] \approx_e col(T,c)[i+1..j]$ for each sampled col c in $O(dn^2)$ time. We treat them as candidates of the summation relationships for T, and then verify whether each candidate satisfies the summation relationship for other cols. Let M be a set of such candidates. For each element of M, verification can be done in $O(m)$ time when C, described in Section 5.1, has been preliminary computed in $O(nm)$. In total, the sample algorithm extracts the summation relationships of T in $O(dn^2 + nm + |M|m)$ time. Though $|M|$ can be $O(n^2)$ in the worst case, it is small in many cases, and the algorithm practically runs faster than the naive algorithm. The sample algorithm may overlook some summation relationships; however, this does not occur often and extraction accuracy becomes relatively better than the naive algorithm for real spreadsheets (see Section 6). This is because the sample algorithm can avoid extracting undesired summation relationships when the input spreadsheet is sparse and most values are zero and the summation relationship accidentally satisfies at few cols.

5.3 Non-negative Algorithm

The non-negative algorithm for a table with only non-negative values is very similar to the sample algorithm. The difference is that the non-negative algorithm extracts the candidate set M of summation relationships which is satisfied on sampled d cols in $O(dn \log n)$ time. An important observation is that for a col c and a table C, preliminary described in Section 5.1, $col(C,c)$ is arranged in non decreasing order since T contains only non-negative values. This implies that for a position i, the range of positions j satisfying $col(T,c)[i] \approx_e col(T,c)[i+1..j]$ are contiguous, and such j can be found by the binary search on $col(C,c)$ in $O(\log n)$ time. Therefore the candidate set M of summation relationships can be obtained by computing the summation relationships for all n positions on each of sampled d cols in $O(dn \log n)$ time. In total, the non-negative algorithm extracts the summation relationships of T in $O(dn \log n + nm + |M|m)$ time in total.

6. EXPERIMENTS

We evaluated the extraction accuracy and runtime performances of our naive, sample, and non-negative algorithms described in Section 5.

All experiments involved spreadsheets that are publically available in PublicData.eu. For ease of evaluation, we used only spreadsheets that include no values represented by for-

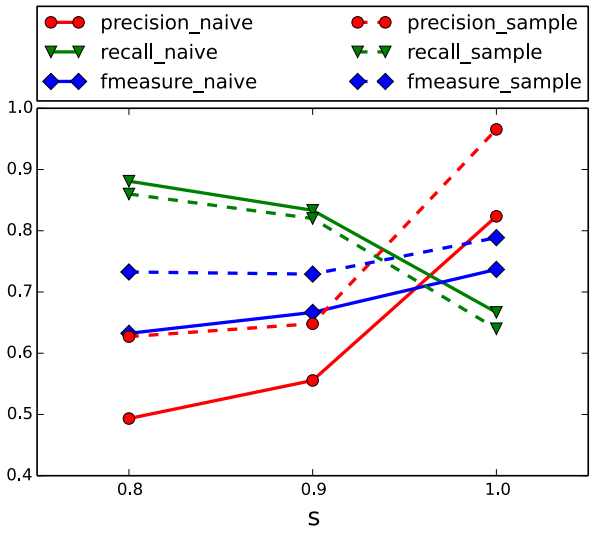

Figure 3: Precision, recall, and F-measure for naive and sample algorithms when e and d were fixed at 0.01 and 3, respectively, and s was varied; 0.8, 0.9, and 1.0.

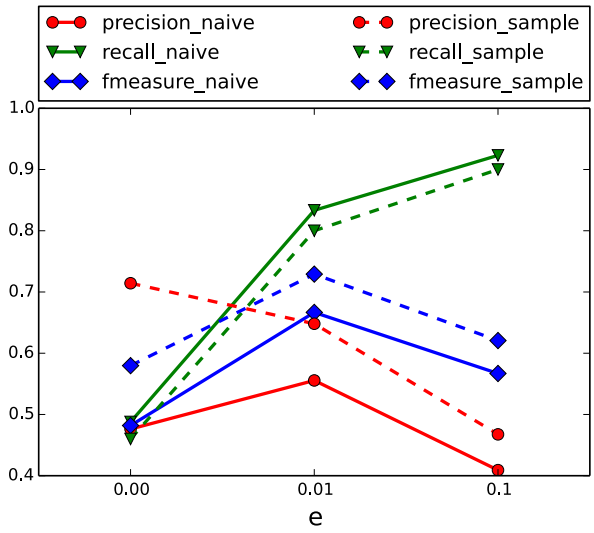

Figure 4: Precision, recall, and F-measure for naive and sample algorithms when s and d were fixed at 0.90 and 3, respectively, and e was varied; 0.00, 0.01, and 0.10.

mulas or references to other values [4] .

6.1 Extraction Accuracy

We evaluated the extraction accuracy of the naive and sample algorithms by comparing the hierarchical summation relationships from these algorithms with the manually extracted one for 50 spreadsheets randomly chosen from PublicData.eu. Since the non-negative algorithm outputs the same result as the sample algorithm when the input spreadsheet contains only non-negative values, we do not include it in the comparison. We considered as the baseline for the evaluation that the summation relationship satisfies $s = 0.8$ and $e = 0.1$ and the concept of a parent header is divided into more specific concepts of child headers. Each relationship, a pair of parent row and its child rows, is treated as a unit, and the hierarchical summation relationship from an algorithm is evaluated by counting the number of units that exactly match with those of the baseline extracted manually.

Figure 3 shows the precision, recall, and F-measure of the naive and sample algorithms when e and d were fixed at 0.01 and 3 and s was varied; 0.8, 0.9, and 1.0. When s increased, the restriction that the summation relationship is satisfied became more strict, so precision tended to increase. On the other hand, recall tended to decrease. The recall at $s = 1.0$ was worse than that at $s = 0.9$. This means that real spreadsheets have a summation relationship that is satisfied for most cols, but a few cols do not, as we discussed in Section 3.3.

Figure 4 shows precision, recall, and F-measure of the naive and sample algorithms when s and d were fixed at 0.9 and 3 and e was varied; 0.00, 0.10, and 0.01. In the same way as Figure 3, when e decreased, the restriction became more strict, so precision tended to increase. On the other hand, the recall tended to decrease.

In our experiments, when s and e were 0.9 and 0.01 respectively, the F-measure was highest for both the naive and sample algorithms. Moreover, the F-measure of sample algorithm (0.73) was slightly better than that of the naive algorithm (0.67). We believe this is because the sample algorithm can avoid extracting undesired summation relationships when the input spreadsheet is sparse and most values are zero and the relationship is accidentally satisfied by a few cols.

6.2 Runtime Performance

We used 1000 spreadsheets randomly chosen from PublicData.eu to evaluate the runtime performance for our algorithms. Figure 5 shows the distribution of their rows. The number of spreadsheets of up to 100 rows is 566(56.6%), and that of up to 10^5 rows is 995(99.5%). The number of spreadsheets of up to 10^5 rows with only non-negative values is 779(77.9%). Figure 6 shows the runtimes of naive, sample, non-negative, and *sample + non-negative* algorithms for spreadsheets of up to 10, 10^2, 10^3, 10^4, 10^5 rows, where sample + non-negative algorithm is the combination of sample and non-negative algorithms that uses non-negative algorithm if an input spreadsheet including only non-negative values and sample algorithm otherwise, and e, s, and d were fixed at 0.01, 0.9, and 3, respectively. The sample + non-negative algorithm runs the fastest, and the non-negative algorithms runs the fastest if an input spreadsheet including only non-negative values. The sample + non-negative algorithm suit to extract hierarchical structures from massive spreadsheets since its runtime is practically fast so that it can process 566(56.6%) spreadsheets of up to 10^2 rows within 28 seconds, and 995(99.5%) spreadsheets of up to 10^5 rows within 16200 seconds.

7. CONCLUSION

[4] Our algorithms can be applied to also spreadsheets including formulas or references by preliminary replacing them with real values. Though there are libraries supporting that such as Apache POI(`https://poi.apache.org/`) or xlrd(`https://github.com/python-excel/xlrd`), they do not support all formats of references and formulas.

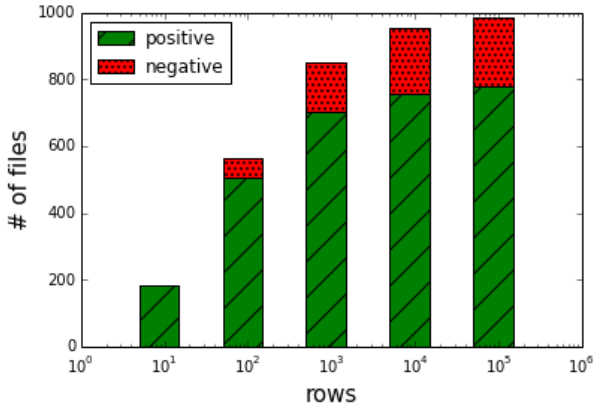

Figure 5: The number of spreadsheets randomly chosen from PublicData.eu of under $10, 10^2, 10^3, 10^4, 10^5$ rows.

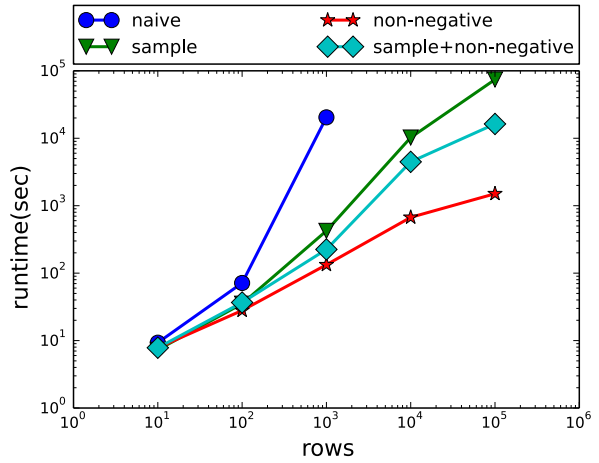

Figure 6: Times for extracting the hierarchical summation relationship for spreadsheets of under $10, 10^2, 10^3, 10^4, 10^5$ rows.

We proposed three algorithms to extract hierarchical structures from spreadsheets. The algorithms rely on the relation of values in which their sum of values approximately equals their preceding or succeding value, which we call *summation relationship*. Therefore, we can apply these algorithms to heterogeneous spreadsheets whose structures are not specified with fonts, space, indent, or keywords. Our experiments showed that our algorithms can be applied to real spreadsheets and their runtimes are practical so that they can be applied to massive spreadsheets.

A future work is to develop methods that can be applied to more heterogeneous spreadsheets with high extraction accuracy. We believe that the combination of our approach, which uses the relation of values, and the approach, which uses the header features, has a potential to be a such better method.

References

[1] C. Bizer, T. Heath, and T. Berners-Lee. Linked data - the story so far. *Int. J. Semantic Web Inf. Syst.*, 5(3):1–22, 2009.

[2] M. J. Cafarella, A. Y. Halevy, and N. Khoussainova. Data integration for the relational web. *PVLDB*, 2(1):1090–1101, 2009.

[3] Z. Chen and M. J. Cafarella. Automatic web spreadsheet data extraction. In *3RD International Workshop on Semantic Search over the Web, SSW '13, Riva del Garda, Italy, August 30, 2013*, pages 1:1–1:8, 2013.

[4] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[5] I. Ermilov, S. Auer, and C. Stadler. Csv2rdf: User-driven csv to rdf mass conversion framework. In *Proceedings of the ISEM '13, September 04 - 06 2013, Graz, Austria*, 2013.

[6] O. Hartig, C. Bizer, and J.-C. Freytag. Executing sparql queries over the web of linked data. In *Proceedings of the 8th International Semantic Web Conference*, ISWC '09, pages 293–309, Berlin, Heidelberg, 2009. Springer-Verlag.

[7] V. Hung, B. Benatallah, and R. Saint-Paul. Spreadsheet-based complex data transformation. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, CIKM '11, pages 1749–1754, New York, NY, USA, 2011. ACM.

[8] T. Igarashi, J. D. Mackinlay, B. Chang, and P. Zellweger. Fluid visualization for spreadsheet structures. In *Proceedings 1998 IEEE Symposium on Visual Languages, Nova Scotia, Canada, September 1-4, 1998*, pages 118–125, 1998.

[9] J. Madhavan, P. A. Bernstein, A. Doan, and A. Y. Halevy. Corpus-based schema matching. In *Proceedings of the 21st International Conference on Data Engineering, ICDE 2005, 5-8 April 2005, Tokyo, Japan*, pages 57–68, 2005.

[10] V. Raman and J. M. Hellerstein. Potter's wheel: An interactive data cleaning system. In *Proceedings of the 27th International Conference on Very Large Data Bases*, VLDB '01, pages 381–390, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.