

# A Clustering Technique for Mining Data from Text Tables\*

Hasan Davulcu<sup>1</sup>, Saikat Mukherjee<sup>1,2</sup>, I.V. Ramakrishnan<sup>1,2</sup>

<sup>1</sup> XSB, Inc.  
Suite 115, Nassau Hall  
Stony Brook, NY 11790, USA

<sup>2</sup> Dept. of Computer Science  
SUNY Stony Brook  
Stony Brook, NY 11794, USA

## Abstract

Considerable quantities of valuable data about product information and financial statements is often available in sources and formats that are not amenable for querying using traditional database techniques. One such important source is text documents. In such documents these kinds of data often appear in tabular form. A data item in these *text tables* may span several words (e.g. product description). Furthermore items supposedly within the same column do not necessarily begin or end at the same position. Thus the absence of any regularity in column separators makes it difficult to automatically *mine*, i.e. extract data items from text tables. Nevertheless an interesting characteristic often exhibited by these tables is that intra-column items are “*closer*” to each other than inter-column items. We exploit this observation to develop a *clustering-based* technique to extract data items from these tables. In contrast to previous approaches, a unique and important aspect of using clustering is that it makes the technique robust in the presence of misalignments. We provide a characterization theorem for text tables on which this technique will always produce a correct extraction. We discuss the design and implementation of a system for extracting tabular data based on this clustering technique. We present experimental evidence of its effectiveness and usability on real industrial data.

**Keywords:** clustering, tables, data extraction, text

Contact Author: Saikat Mukherjee  
E-mail: [saikat@cs.sunysb.edu](mailto:saikat@cs.sunysb.edu)

---

\*This work was supported by industry and university grant – NSF IIS0072927

# 1 Introduction

A wealth of information relevant for electronic commerce often appears in *text* form. This includes specification and performance data sheets of products created by manufacturers, financial statements published by brokerage houses, product offerings by vendors, etc. Typically these types of product and financial data are published in tabular form. Figure 1 is an illustrative example of a financial statement presented as a table<sup>1</sup> in text form, which we will refer to as the *text table*.

```

12345678901234567890123456789012345678901234567890123456789012345678901234567890
1 SECURITIES                                RATE    MATURITY    AMOUNT    BASIS
2 -----                                - - - - -  - - - - -  - - - - -  - - - - -
3
4 Carson City Nev Sch Dist.....           7.700    2001        265,000    267,100
5 Trans Authority in the state of Delaware... 7.100    2001         200,000    201,981
6 Harrison Cnty MS Sch Dist.....           7.000    2002         70,000     72,19
7 NYS Environ.Fac Corp.....              7.300    2002         500,000    507,985
8 Lewisville TX Indpt Sch Dist.....         7.500    2002         300,000     3,137
9 Hamilton Twp NJ Sch Dist.....            7.000    2002         260,000    270,868
0 Lincoln, MI Cons Sch Dist.....           7.000    2003         325,000    345,241
1 Tempe, AZ.....                          6325     2003         210,000    221,770
2 Rowlett, TX.....                        6.000     2004         170,000    177,535
3 Gladstone MI Pub Schedule in Michigan USA... 6.500     2005         100,000    109,035
4 NYS Dorm Auth.....                      6.500     2005         250,000     26,806
5 FL St Brd Mand SKG.....                 7.500     2005          20,000     20,644
6 Metro Pier and Expo.....                6.250     2005         100,000    108,603
7 Metro Pier and Exploration of oil and gas... 6.500     2005         310,000     3,373
8 Jackson Miss Pub Sch.....               6.250     2005         130,000    137,765
9 NY Dorm Auth.....                       7.800     2005          40,000    41,610

```

Figure 1: A Finance Data Table

One would like to query the information available in text tables as in Figure 1 above. However this information which is in unstructured text form, is not readily queriable using traditional database technology such as SQL. One way to make it amenable to standard database querying techniques is to *mine*, i.e. *extract* the data items in the tables and populate a database out of the extracted data. But extracting text data from tables that are “*irregular*” as in the example above poses some difficulties. In broad terms, irregularity is characterized by variable length data items (perhaps spanning multiple words) that possibly overlap with items in neighboring columns. For example in Figure 1 the item *Gladstone MI Pub Schedule in Michigan USA* in the first column overlaps with items (7.700, 7.000 and 7.000 from the 1st, 2nd and 7th rows respectively) in the second column. In fact this table was one of over several thousand text tables sent to us by a financial data aggregation company that had tasked us to develop an extractor for these tables.

Let us now examine some possible approaches to extraction from such tables. Assume that every character (including white spaces) in a row is assigned a unique position in that row. Since an item in the table is a string, we can further suppose that every item  $i$  in a row is associated with a pair  $\langle l_i, r_i \rangle$  where  $l_i, r_i$  are the positions where the item begins and ends respectively. A simple approach for extracting items is to find *fixed separators* between successive columns. Intuitively, a fixed separator

<sup>1</sup>The topmost row denotes the column position of each character and the first column in each row denotes the row number in the table.

is a unique position that distinguishes items occurring in a pair of neighboring columns. In particular if  $p$  is a fixed separator for columns  $k$  and  $k + 1$  then  $\forall$  items  $i \in k$ ,  $r_i < p$  and  $\forall$  items  $j \in k + 1$ ,  $l_j > p$ , i.e. all the items in column  $k$  occur before  $p$  while those in  $k + 1$  occur after  $p$ . In fact a recent work [1] uses such an approach for extracting data from tables. But observe in Figure 1 that we cannot always find fixed separators (as in columns 1 and 2 in Figure 1). Even if fixed separators exist it is unclear how they can unambiguously separate columns that have multiword items (e.g. Column 1 in Figure 1). Another technique that is generally used for extracting data from text is based on regular expressions [6]. Regular expressions specify patterns that occur in text and a regular expression processing engine extracts pieces of text that match the specified patterns [4]. Although regular expression based extractors are powerful when dealing with text processing in general, they are quite cumbersome and difficult to use in the presence of tables consisting of items that span several words and overlap with items in other columns. The problem we address in this paper is how do we extract data from text tables such as those in Figure 1.

**Overview of Approach and Summary of Results** Observe in Figure 1 that although fixed separators between every pair of adjacent columns do not exist, by visual inspection a casual observer can still correctly associate every item to its corresponding column. This is because all the items belonging to a column, despite having irregular alignments, appear clustered more “*closely*” to each other than to items in different columns. Whereas such clusters can be clearly discerned by a human observer, making them machine recognizable is the key to automated extraction of data items from text tables.

- In this paper we develop a *clustering* technique for extracting items from irregular text tables. We assume that the tables have headers corresponding to each column. We first formalize the notion of “closeness” of data items. Using this notion we develop an iterative algorithm to partition all the data items into clusters and associate each cluster with a distinct header.
- We also formalize the notion of a *correct* extraction for a table and provide a *syntactic* characterization theorem for tables on which our algorithm will always produce a correct extraction (see Section 2).
- We exploit the characterization to engineer a practical system that can be configured to produce a high extraction yield as well as facilitate identification of incorrect extractions (see Section 3).

## Research Contributions

- A number of clustering algorithms have been developed in the past for applications in image processing, machine learning and data mining [9, 3]. We use clustering for the purpose of extracting all the column items from a text table. To the best of our knowledge applying a clustering technique to this problem has not yet been explored in the research literature.
- Clustering enables us to make associations between items in a column based not merely on examining items in adjacent rows but across all the rows in the table. This means that even though two items in adjacent rows may not appear to be in the same column when examined in isolation (due to misalignments), but when viewed in the context of all the rows they can be correctly associated with the same column. In contrast to previous techniques [1, 8, 5, 6, 7] a clustering-based algorithm is robust in the presence of misalignments.
- We have developed the first formal treatment of the problem. The formalization yielded the characterization theorem which from a theoretical perspective, can serve as the basis for comparing

the power of different approaches. On the practical side it can provide valuable information for improving extraction yield by examining the incorrectly extracted tables and appropriately reconfiguring and rerunning the system on these tables (see Section 3).

The rest of the paper is organized as follows. In Section 2 we develop the concept of clustering pertinent to the problem of table extraction as well as the notion of a correct extraction. Based on this clustering concept, we describe an algorithm to extract items from a table. Characterization of tables on which this algorithm always yields a correct extraction also appears in this section. In Section 3 we describe an implementation of our extraction system based on clustering and present experimental evidence of its effectiveness on real industrial data. Our algorithm assumes that every column is associated with a unique header. However these headers may span multiple lines. We briefly discuss how to detect such headers and uniquely associate them with their corresponding columns. Related work appears in Section 4. Discussions and concluding remarks appear in Section 5 where we mention how similar clustering ideas can be extended for *detecting tables* embedded in arbitrary text documents.

## 2 A Clustering Technique for Table Extraction

The problem we address is this: *Given the rows of a table embedded in text as the input, create an algorithm that associates the items in the table with their corresponding columns.*

We develop a clustering-based algorithm for the above problem. We will require the concept of a cluster appropriate for the above problem. Also observe that the result produced by the algorithm is an association between the items in the table with columns. We therefore need the notion of a correct association to evaluate its output. To formalize all these concepts we first develop the technical machinery required to describe the algorithm and its properties.

### 2.1 Formalization

We assume that a line in a text table is made up of characters. Each character in a line is associated with a unique position. We will use lines and rows interchangeably. Each row has a unique integer index called its *row index*. A line is made up of *tokens* defined as:

**Definition 1 (token)** : *A token is a contiguous sequence of characters until either a blank space or a newline character is encountered.*

A token  $t$  is characterized as a four tuple  $(start, end, center, row)$  where :

- $start(t)$  is the position of its first character.
- $end(t)$  is the position of its last character.
- $center(t) = \lceil start + end \rceil / 2$ .
- $row(t)$  is the index of the row in which it occurs.

In Figure 1, the item “*Delaware...*” in row 5 is a token whose start, end, center and row index are 35, 46, 41 and 5 respectively.

**Definition 2 (cluster)** : *Let  $S$  be a set of tokens.  $S$  is a cluster if  $\forall t_k \in S \exists t'_k \in S$  such that:  $\forall t''_k \notin S$   $|center(t_k) - center(t'_k)| \leq |center(t_k) - center(t''_k)|$ .*

We can partition the entire set of tokens into clusters as shown in Figure 2. In the figure, tokens  $t_1, t_4, t_5$  and  $t_6$  belong to cluster  $C_1$  whereas tokens  $t_2, t_3, t_7$  and  $t_8$  belong to cluster  $C_2$ . It can be verified that the two clusters satisfy definition 2.

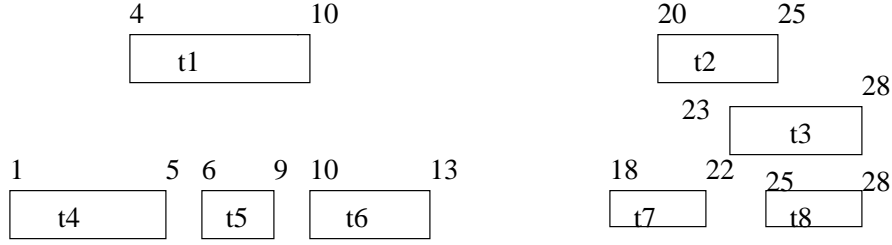


Figure 2: Example of Clusters

Observe in Figure 2 that the positions of tokens in  $C_1$  precede those in  $C_2$ . We can hence impose a linear order on the set of all clusters in a partition. This ordering will be based on certain “extremal” tokens in the clusters. We now set up the concepts necessary to define this order. First observe in Figure 2 that we can identify certain tokens as being at the boundaries of a cluster.

In the following definitions  $t$ ,  $t_l$  and  $t_r$  are tokens belonging to the same cluster  $C_i$ .

**Definition 3 (boundary tokens)** : Let  $C_i$  denote a cluster and  $j$  be some row. Then,

$$\text{border}(i, j) = \begin{cases} \text{undefined}, & \text{if there are no tokens in the row.} \\ t, & \text{if there is only one token } t \text{ in the row.} \\ t_l, t_r, & \text{s.t. } \nexists \text{ tokens } t'_l, t'_r \in C_i \wedge \text{row } j \text{ with } \text{start}(t_l) > \text{start}(t'_l) \wedge \text{end}(t_r) < \text{end}(t'_r). \end{cases}$$

In Figure 2, the boundary tokens for  $C_1$  are  $t_1, t_4$  and  $t_6$ . We can classify all the tokens in a table unambiguously as either boundary or non-boundary tokens. Since every token in a row has unique start and end positions we can define the rightmost and leftmost token of a row in a cluster as:

**Definition 4** Let  $C_i$  denote a cluster and  $j$  denote a row index. Then,

$$\text{rightmost}(i, j) = \begin{cases} \text{undefined}, & \nexists \text{ a border token } t \text{ with } \text{row}(t) = j. \\ t, & t \in \text{border}(i, j) \wedge \nexists t' \in \text{border}(i, j) \text{ s.t. } \text{end}(t') > \text{end}(t). \end{cases}$$

$$\text{leftmost}(i, j) = \begin{cases} \text{undefined}, & \nexists \text{ a border token } t \text{ with } \text{row}(t) = j. \\ t, & t \in \text{border}(i, j) \wedge \nexists t' \in \text{border}(i, j) \text{ s.t. } \text{start}(t') < \text{start}(t). \end{cases}$$

For cluster  $C_1$  in Figure 2, the rightmost (leftmost) tokens are  $t_1$  ( $t_1$ ) for the first row, is undefined for the second row and is  $t_6$  ( $t_4$ ) for the third row. Note, the rightmost( $i, j$ ) or the leftmost( $i, j$ ) may not be the spatially rightmost token or leftmost token for the  $j^{\text{th}}$  row of  $C_i$ . The additional requirement is that they be border tokens. We now define the extremal tokens of a cluster.

**Definition 5 (extremal tokens)** : For a cluster  $C_i$  the rightmost token is defined as:

$$\begin{aligned} \text{rightmost}(i) &= \text{rightmost}(i, j) \wedge \nexists j' \text{ s.t. } \text{rightmost}(i, j') \wedge \text{end}(\text{rightmost}(i, j)) < \text{end}(\text{rightmost}(i, j')). \\ \text{leftmost}(i) &= \text{leftmost}(i, j) \wedge \nexists j' \text{ s.t. } \text{leftmost}(i, j') \wedge \text{start}(\text{leftmost}(i, j)) > \text{start}(\text{leftmost}(i, j')). \end{aligned}$$

In Figure 2,  $t_6$  is the rightmost token of  $C_1$  and  $t_4$  is its leftmost token. We can pick one of the extremal tokens consistently (say the leftmost) from every cluster and use their start positions to

linearly order all the clusters in the partition. We will henceforth assume that tokens are partitioned into clusters  $C_1, C_2, \dots, C_n$  where if  $i < j$  then  $C_i$  occurs before  $C_j$  in the order. In Figure 2 we can pick the leftmost tokens in the two clusters and use their start positions to place  $C_1$  before  $C_2$ .

We say that  $C_i$  and  $C_j$  are adjacent if there is no other cluster between them. The span of a cluster  $C_i$ , denoted  $span(C_i)$ , is the sequence of positions  $\langle start(leftmost(i)), start(leftmost(i)) + 1, \dots, end(rightmost(i)) \rangle$ . In Figure 2,  $span(C_1)$  is  $\langle 1, 2, \dots, 13 \rangle$ .

We are now ready to define the important notion of ‘‘closeness’’ between clusters. Our notion is based on ‘‘averaging’’ the length of inter-cluster gaps between adjacent clusters using their boundary tokens.

We associate a gap between  $C_i$  and  $C_{i-1}$ , denoted  $gap\_on\_left(i)$  and a gap between  $C_i$  and  $C_{i+1}$ , denoted as  $gap\_on\_right(i)$ . The  $gap\_on\_right$  and the  $gap\_on\_left$  are defined in terms of intrarow gaps as follows.

**Definition 6 (intrarow\_sep)** *The gap between cluster  $C_i$  and  $C_{i+1}$  in the  $j^{th}$  row, denoted,*

$$intrarow\_sep_r(i, j) = \begin{cases} start(leftmost(i+1, j)) - end(rightmost(i, j)), & \text{if both } leftmost(i+1, j) \text{ and } rightmost(i, j) \text{ are defined.} \\ \max(0, start(leftmost(i+1, j)) - end(rightmost(i))), & \text{if } rightmost(i, j) \text{ is undefined but } leftmost(i+1, j) \text{ is defined.} \\ \max(0, start(leftmost(i+1)) - end(rightmost(i, j))), & \text{if } leftmost(i+1, j) \text{ is undefined but } rightmost(i, j) \text{ is defined.} \\ 0, & \text{if both are undefined.} \end{cases}$$

*The gap between cluster  $C_i$  and  $C_{i-1}$  in the  $j^{th}$  row, denoted,*

$$intrarow\_sep_l(i, j) = \begin{cases} start(leftmost(i, j)) - end(rightmost(i-1, j)), & \text{if both } leftmost(i, j) \text{ and } rightmost(i-1, j) \text{ are defined.} \\ \max(0, start(leftmost(i, j)) - end(rightmost(i-1))), & \text{if } rightmost(i-1, j) \text{ is undefined but } leftmost(i, j) \text{ is defined.} \\ \max(0, start(leftmost(i)) - end(rightmost(i-1, j))), & \text{if } leftmost(i, j) \text{ is undefined but } rightmost(i-1, j) \text{ is defined.} \\ 0, & \text{if both are undefined.} \end{cases}$$

Informally, the intrarow gap between  $C_i$  and  $C_{i+1}$  refers to the gap between a pair of extremal tokens that occur in the same row. We have to take into account the cases where both tokens exist in both the clusters, only one of them exists in one of the clusters and neither exist. In the last case the gap is considered to be undefined. In the case when only one of them exists the closest token is chosen. This choice is made to ensure that the gap is always minimal. Note that no gaps exist between overlapping tokens.

**Definition 7 (gaps)**  $gap\_on\_right(i)$  is  $(\sum row\_gap\_on\_right(i, j)) / M$ , where  $M$  is the number of rows of the cluster where  $row\_gap\_on\_right(i, j) > 0$ . Similarly,  $gap\_on\_left(i)$  is  $(\sum row\_gap\_on\_left(i, j)) / M$ .

In Figure 2, let us consider the three clusters  $C_1$  consisting of the token  $t_4$ ,  $C_2$  consisting of the tokens  $t_1$  and  $t_5$  and  $C_3$  consisting of the token  $t_6$ . For  $C_2$ , the  $intrarow\_sep_l(2, 1) = \max(0, start(t_1) - end(t_4)) = 0$ , the  $intrarow\_sep_l(2, 2) = 0$  (both undefined) and  $intrarow\_sep_l(2, 3) = start(t_5) - end(t_4) = 1$ . Thus, the average inter cluster gap between  $C_1$  and  $C_2$ ,  $gap\_on\_left(2) = intrarow\_sep_l(2, 3) / 1 = 1$ . The  $gap\_on\_right$  for any cluster is also calculated in a similar way.

## 2.2 Building blocks

We sketch a high-level overview of our clustering algorithm. It has very close parallels to iterative partition refinement algorithms. Starting with an initial partitioning of the set of tokens into clusters, the partition gets refined in every iteration. Refinement amounts to creating larger clusters by merging adjacent ones based on inter-cluster gaps. The algorithm terminates when no more refinement is possible.

We will now construct the essential building blocks for our algorithm, namely, creating the initial partition, merging of clusters in an iteration and termination.

**Initial partitioning** The initial partition puts all tokens with a high-degree of overlap into one cluster. Intuitively what this means is that all such tokens belong to the same column. Formally:

**Definition 8 (overlapping tokens)** : *Tokens  $t_i$  and  $t_j$  overlap whenever one of the following holds:*

1.  $start(t_i) \leq start(t_j) \leq end(t_i)$ .
2.  $start(t_i) \leq end(t_j) \leq end(t_i)$ .
3.  $start(t_j) \leq start(t_i)$  and  $end(t_j) \geq end(t_i)$

In Figure 2, tokens  $t_2$  and  $t_3$  overlap. We use this notion to define:

**Definition 9 (high degree of overlap between tokens)** : *Tokens  $t_i$  and  $t_j$ , with centers  $c_i$  and  $c_j$  respectively, have a high degree of overlap whenever:*

$$c_i = \begin{cases} c_j \\ c_j + 1 \\ c_j - 1 \end{cases}$$

In Figure 2, tokens  $t_1$  and  $t_5$  have a high degree of overlap between them since  $center(t_1) = 7$  and  $center(t_5) = 8$ . From definition 9 it is easy to see that the centers of tokens with a high-degree of overlap occupy consecutive positions. To create the initial set of clusters, we add tokens whose centers are on consecutive positions to the same set. It is easy to see that each such set is a cluster.

**Merging clusters** We say that adjacent clusters  $C_i$  and  $C_{i-1}$  are “*mutually close*” whenever  $(gap\_on\_left(i) < gap\_on\_right(i)) \wedge (gap\_on\_right(i - 1) < gap\_on\_left(i - 1))$ . We only merge mutually close clusters into a larger cluster. Once we merge the pair of clusters into a bigger cluster we will have to associate gaps with this bigger cluster. If  $C_i$  is merged with  $C_{i-1}$  into the cluster  $C$  then  $gap\_on\_left(i-1)$  and  $gap\_on\_right(i)$  become  $C$ ’s left and right gaps respectively. On the other hand if  $C_i$  is merged with  $C_{i+1}$  then  $gap\_on\_left(i)$  and  $gap\_on\_right(i+1)$  become left and right gaps respectively of cluster  $C$ .

However even when adjacent clusters are mutually close to each other we do not merge them if they belong to different columns. How do we identify such a case since we are not provided with any information about columns? We use headers for this purpose. We assume that every column is associated with a distinct header string. The span of a header, analogous to that of a cluster, is the sequence of the positions associated with the characters in its string. In Figure 1, the span of the header “SECURITIES” is  $\langle 3, 4, 5, \dots, 12 \rangle$ . We say that a cluster is distinctly associated with a header when their spans overlap. We do not merge mutually close clusters whenever they are uniquely associated with their headers. In such a case we say that the clusters belong to different columns.

**Termination** Note that in a iteration we may fail to merge a pair of adjacent clusters because:

1. They are not mutually close.
2. They belong to different columns.

If in an iteration we are unable to merge any pair of clusters the algorithm terminates.

### 2.3 Putting it All Together

We present here Algorithm `Extract_Columns` using the building blocks described above. `Extract_Columns` scans the rows of the table as it's input. It will produce as the result an association of the items in the table with columns. Recall that two adjacent clusters are merged when they are mutually close. To facilitate merger we maintain a variable *link* with each cluster  $C_i$ . During the iteration,  $\text{link}(C_i)$  points to  $C_{i-1}$  ( $C_{i+1}$ ) if  $C_i$  is closer to  $C_{i-1}$  ( $C_{i+1}$ ). In the algorithm, we use  $G_i$  to denote the gap between  $C_i$  and  $C_{i+1}$  i.e.  $G_i = \text{gap\_on\_right}(C_i) = \text{gap\_on\_left}(C_{i+1})$ . The procedure `Initial_Clusters` creates the initial partition of tokens into a set of clusters. After that clusters overlapping with the same header are merged together. The procedure `Inter_Cluster_Gaps` computes the gaps between adjacent clusters using the initial partitions. The procedure `Merge_Clusters` merges mutually close clusters subject to the condition that their spans do not overlap with two different headers. The merged cluster produced contains the union of the tokens of the two clusters and if either of them is associated with a header then the resulting cluster is also associated with the same header.

For example, in Figure 2 the initial set of clusters are  $C_1 = t_4$ ,  $C_2 = t_1, t_5$ ,  $C_3 = t_6$ ,  $C_4 = t_7$ ,  $C_5 = t_2$ ,  $C_6 = t_3, t_8$ . Thus,  $G_1 = 1$ ,  $G_2 = 1$ ,  $G_3 = 5$ ,  $G_4 = 0$  and  $G_5 = 0$ . In the first iteration,  $\text{link}(C_1) = C_2$ ,  $\text{link}(C_2) = C_1$ ,  $\text{link}(C_3) = C_2$ ,  $\text{link}(C_4) = C_5$ ,  $\text{link}(C_5) = C_4$  and  $\text{link}(C_6) = C_5$ . So we merge clusters  $C_1$  and  $C_2$  and clusters  $C_4$  and  $C_5$ . Our new set of clusters are  $C'_1 = \text{merge}(C_1, C_2)$ ,  $C'_2 = C_3$ ,  $C'_3 = \text{merge}(C_4, C_5)$  and  $C'_4 = C_6$ . Our new set of intercluster gaps are  $G'_1 = G_2$ ,  $G'_2 = G_3$ ,  $G'_3 = G_5$ . In the second iteration,  $\text{link}(C'_1) = C'_2$ ,  $\text{link}(C'_2) = C'_1$ ,  $\text{link}(C'_3) = C'_4$  and  $\text{link}(C'_4) = C'_3$ . Thus, we merge clusters  $C'_1$  and  $C'_2$  and clusters  $C'_3$  and  $C'_4$ . Our new set of clusters are  $C''_1 = \text{merge}(C'_1, C'_2)$  and  $C''_2 = \text{merge}(C'_3, C'_4)$ . Our new intercluster gap is  $G''_1 = G_3$ . In the third iteration we reach a fixpoint since we cannot merge these two clusters as they overlap with two different headers. Thus, the algorithm reaches a fixpoint and gives the correct set of clusters.

**Algorithm**ExtractColumns**begin**

```

1. Invoke Initial_Clusters to form the set of initial clusters  $\prod_C = \{C_1, C_2, \dots, C_n\}$ 
2. Invoke Inter_Cluster_Gaps to compute the gaps between the initial clusters,  $Gap = \{G_1, G_2, \dots, G_{n-1}\}$ 
3. while( $|\prod_C| \neq |\prod_{C'}|$ ) do (* a fixpoint check is done *)
4.    $\prod_C = \prod_{C'}$ 
5.    $Gap = Gap'$ 
6.   forall  $i, 1 \leq i \leq |\prod_C|$  do
7.      $link(C_i) = nil$ 
8.   end
9.   forall  $i, 1 \leq i \leq |\prod_C| - 2$  do (* for each cluster the gaps between *)
10.    if( $G_i \leq G_{i+1}$ ) then (* it's neighbors are compared *)
11.       $link(C_{i+1}) = C_i$  (* and the closeness of that cluster is determined *)
12.    else
13.       $link(C_{i+1}) = C_{i+2}$ 
14.    endif
15.  end
16.   $link(C_1) = C_2$ 
17.   $link(C_{|\prod_C|}) = C_{|\prod_C|-1}$ 
18.   $j = 1$ 
19.  forall  $i, 1 \leq i \leq |\prod_C| - 1$  do (* mutually close clusters are merged *)
20.    if( $link(C_i) = right \ \& \ link(C_{i+1}) = left$ ) then (* together to form a single cluster *)
21.       $C_j = Merge\_Clusters(C_i, C_{i+1})$  (* if they do not overlap with different headers *)
22.       $G_j = G_{i+1}$ 
23.       $i = i + 1$ 
24.    else
25.       $C_j = C_i$ 
26.       $G_j = G_{i+1}$ 
27.    endif
28.     $\prod_{C'} = \prod_{C'} \cup C_j$  (* the new set of clusters are created *)
29.     $Gap' = Gap' \cup G_j$  (* gaps are associated with this new set *)
30.     $j = j + 1$ 
31.  end
32. endwhile
33. return $\prod_C$ 
end

```

## 2.4 Algorithmic Properties

The objective of the underlying algorithm *ExtractColumns* is to correctly associate items with a column. Can it always do so? Herein we address the question of a *correct* extraction of columns w.r.t. a table whose columns have been already demarcated by a user. This demarcation represents the “correct extraction” expected by the user. The formalization proceeds as follows.

The span of a column is the contiguous sequence of positions from the start of it’s leftmost token to the end of it’s rightmost token. One can conceptually think of the span as two imaginary lines such that the span of no token in the column extends beyond these two lines. Span of a column is a purely visual concept. Using columns we define:

**Definition 10 (table)** : *A table is a partition of the tokens into a set of columns  $Col_1, Col_2, \dots, Col_n$  such that there is a distinct header associated with every column (i.e. their spans overlap) and a token  $t \in Col_i \iff center(t)$  is contained in the span of  $Col_i$ .*

Note that this definition allows spans of adjacent columns to overlap so long as the centers of the shared tokens belong to only one of the columns. In Figure 1, spans of columns 1,2,3,4 and 5 are  $\langle 3,4,\dots,46 \rangle$ ,  $\langle 45,\dots,54 \rangle$ ,  $\langle 55,\dots,64 \rangle$ ,  $\langle 68,69,\dots,80 \rangle$  and  $\langle 83,84,\dots,92 \rangle$  respectively. The headers “SECURITIES”, “RATE”, “MATURITY”, “AMOUNT” and “BASIS” are associated with Columns 1,2,3,4 and 5 respectively.

In the following, we will assume that  $T$  denotes a table whose columns have been demarcated by the user. Suppose the rows of a table  $T$  are supplied as input to *ExtractColumns* and it produces the clusters  $C_1, C_2, \dots, C_n$  as it’s output upon termination. We say that this extraction is correct w.r.t.  $T$  iff every  $C_i$  is uniquely associated with a header and there is a bijective mapping between the columns of  $T$  and the extracted clusters. We can guarantee the following:

**Theorem 1** *If in every iteration ExtractColumns only merges mutually close clusters belonging to the same column of T then it will always produce a correct extraction of the table T.*

We now characterize tables for which *ExtractColumns* will always yield a correct extraction. We will base this characterization on the gaps between non-overlapping tokens in a column. In the definitions below let  $p, q, q'$  denote tokens.

**Definition 11 (token\_gaps)** *The gap between token  $p$  and some other token  $q$  (both  $p$  and  $q$  in the same row  $j$ ) denoted,*

$$\text{intra\_row\_gap}(p) = \begin{cases} \text{start}(q) - \text{end}(p), & \text{start}(q) > \text{end}(p) \wedge \exists q' \text{ in } j \text{ s.t. } \text{start}(q') < \text{start}(q). \\ \text{undefined}, & \text{otherwise.} \end{cases}$$

*The gap between token  $p$  and some other token  $q$  not in the same row denoted,*

$$\text{inter\_row\_gap}(p) = \begin{cases} \text{start}(q) - \text{end}(p), & \text{start}(q) > \text{end}(p) \wedge \text{intra\_row\_gap}(p) \text{ is undefined} \\ & \wedge \exists q' \text{ s.t. } \text{end}(p) < \text{start}(q') < \text{start}(q). \\ \text{undefined}, & \text{otherwise.} \end{cases}$$

Let  $\text{columngap}(i)$  denote the maximum over all intra\_row\_gaps and inter\_row\_gaps of tokens in column  $i$ . Furthermore, let  $\text{min\_columngap}$  denote the smallest inter column gap.

**Theorem 2 (Characterization of tables)** *For every column  $i$  in  $T$ , if  $\text{columngap}(i) < \text{min\_columngap}$  then ExtractColumns will produce a correct extraction for  $T$ .*

**Proof sketch:** By contradiction. Assume *ExtractColumns* terminates with an incorrect partitioning. This means that in some iteration, a cluster  $C_i$  was merged with  $C_{i+1}$  while it should have been merged with  $C_{i-1}$ . Let the gap between  $C_i$  and  $C_{i-1}$  be  $d_{i-1}$  and the gap between  $C_i$  and  $C_{i+1}$  be  $d_{i+1}$ . Since,  $C_i$  is (incorrectly) merged with  $C_{i+1}$  this means that  $d_{i+1} < d_{i-1}$ . However, since  $C_i$  and  $C_{i-1}$  actually belong to the same column  $d_{i-1} \leq \text{columngap}(i)$ . Also, since in reality  $C_i$  and  $C_{i+1}$  belong to different columns  $d_{i+1} \geq \text{min\_columngap}$ . Using our characterization theorem we can derive,  $d_{i-1} \leq \text{columngap}(i) < \text{min\_columngap} \leq d_{i+1}$ . Thus,  $d_{i-1} < d_{i+1}$  and we get a contradiction. Therefore our assumption is incorrect and we always produce a correct extraction.

For example, in Figure 3, if the user defines table  $T'$  with columns  $C'_1$  and  $C'_2$  then our characterization enables us to decide a priori that *ExtractColumns* will not generate a correct extraction w.r.t.  $T'$ . This is because the gap between tokens  $t_2$  and  $t_5$  (both belonging to column  $C'_1$ ) is more than the gap between  $C'_1$  and  $C'_2$  violating the condition of Theorem 2. If however the user defines table  $T$  with columns  $C_1$  and  $C_2$  then our output will be a correct extraction w.r.t  $T$ .

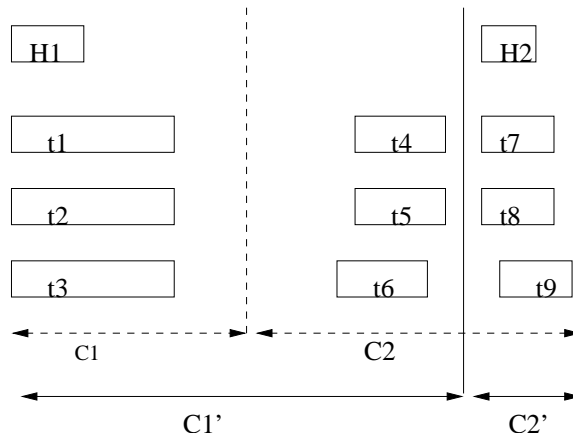


Figure 3: Illustration of Theorem 2

Note that characterization of tables on which the algorithm is guaranteed to produce a correct extraction is based on the column spans. This information can only be supplied externally as it is not computed by our algorithm. So it may appear that such a characterization may not be useful. However we describe how it can be used in practice especially where the tables are generated automatically such as financial and product data. In these cases all of the tables are “structurally similar”. We can hence sample a few tables and determine whether they satisfy our characterization. In such a case we can run the algorithm over all such tables in a batch knowing a priori that the extraction will be correct. In fact this has been validated in our work with the data provided to us by a financial company and also tables containing electronic part descriptions supplied to us by an electronic component catalog manufacturer. We elaborate on this idea in the next section.

### 3 The Extraction System based on Clustering

We implemented a extraction system for text tables based on our clustering algorithm. It is written in Java and the entire system is approximately 3000 lines of code.

The clustering algorithm assumes that each column is associated with a unique header. Each header is a string consisting of one or more words. Typically the text table can be logically separated into two consecutive regions, namely the header region consisting of all the headers followed by the data region. The header region may span multiple lines and the two regions are separated by special tokens which may include blank lines. We assume that the user supplies the list of keywords appearing in the headers as well as the separator tokens. Our system must first discover the headers associated with the columns. This can become quite subtle when the separator tokens are interspersed with the headers. We have developed an algorithm to do header discovery whose details are beyond the scope of this paper.

Below we now describe some of the key features of our system:

**Look and Feel:** The system can be set up to do automatic extraction over a collection of text tables. It has a graphical user interface which is divided into four panels. To use this system, an initial setup prior to extraction is required. This consists of:

1. Loading the input files containing the tables. There is an input panel in the GUI where the user

can load up a set of files each containing a text table or an entire directory containing all the files.

2. Specifying the destination directory where the output tables will be generated. There is an output panel in the GUI where the user can specify this location.
3. The user specifies the headers of columns to be extracted by their header name. We have a panel in our GUI for loading up these header names. The user can provide a list of all the header names that can possibly occur across all tables. The user can also enter phrases as keywords besides providing the union of all the keywords.
4. Tokens which serve as separators between headers and data, like a blank line (newline) or special characters like "\_", "-", etc, are provided as *separator keywords* between the headers and the data. There is a panel where the user enters the separators.

**Error Handling:** During an extraction run the system partitions all the text files into two categories, namely, those it considers have been correctly extracted and those in which errors were detected. It can readily detect certain types of errors. These are:

1. *Header mismatch:* This error occurs whenever either the column corresponding to a header is empty or the configuration parameter is underestimated. The latter will prevent clusters consisting of items belonging to the same column from being merged thereby creating an erroneous number of columns.
2. *Failure of header discovery:* The algorithm for identifying headers cannot detect the boundary separating the header and data regions in the text table.
3. *Absence of any user supplied headers:* The text table does not contain any of the headers supplied by the user.

During a run the system further partitions tables in which errors were detected into one of the above three sub-categories.

**Configuration:** A run of the system amounts to doing extraction over a collection of text tables. For correct extraction we need to know the minimum column gap (see Theorem 2). We discuss now how we can get an estimate of this parameter. The system is used interactively by the user to sample a few text tables and estimate the minimum column gap. The estimated gap becomes a configuration parameter of the system. The algorithm uses this parameter when merging clusters. In particular it will not merge adjacent clusters if the gap between them is larger than the estimated minimum column gap. (The basis for this step is Theorem 2.) The system will use this parameter during a run. It partitions the results of the run into correct and erroneous sub-categories (see “error handling” describe above). At the end of a run the user can examine the text tables on which the extraction failed, identify if it was the caused by an erroneous estimate, collect all such cases, readjust the parameter and start a new run on them. This process can help the user obtain a high extraction yield. Finally we remark that the estimated minimum column gap cannot be used as a fixed separator since this would determine whether data items belonged to the same column based only on examining the items on adjacent rows, making it brittle to misalignments.

	A	B	C	D	E	F	G
1	SECURITIES	RATE	MATURITY	AMOUNT	BASIS		
3	Carson City Nev Sch Dist.....	7.7	2001	265,000	267,100		
4	Trans Authority in the state of Delaware....	7.1	2001	200,000	201,981		
5	Harrison Cnty MS Sch Dist.....	7	2002	70,000	72,19		
6	NYS Environ.Fac Corp.....	7.3	2002	500,000	507,985		
7	Lewisville TX Indpt Sch Dist.....	7.5	2002	300,000	3,137		
8	Hamilton Twp NJ Sch Dist.....	7	2002	260,000	270,868		
9	Lincoln, MI Cons Sch Dist.....	7	2003	325,000	345,241		
10	Tempe, AZ.....	6325	2003	210,000	221,770		
11	Rowlett, TX.....	6	2004	170,000	177,535		
12	Gladstone MI Pub Schedule in Michigan USA...	6.5	2005	100,000	109,035		
13	NYS Dorm Auth.....	6.5	2005	250,000	26,806		
14	FL St Brd Mand SKG.....	7.5	2005	20,000	20,644		
15	Metro Pier and Expo.....	6.25	2005	100,000	108,603		
16	Metro Pier and Exploration of oil and gas....	6.5	2005	310,000	3,373		
17	Jackson Miss Pub Sch.....	6.25	2005	130,000	137,765		
18	NY Dorm Auth.....	7.8	2005	40,000	41,610		
19							
20							
21							
22							
23							
24							
25							

Figure 4: Excel Output of Extract\_Column on the example table in Figure 1

**Performance Measurements** We have tested our system using text tables consisting of financial and electronics parts data. A total of 516 input files, each consisting of a text table, were loaded up and the algorithm was run in batch over this entire collection. Out of these, 426 tables were properly extracted representing a yield of 83% while 74 files suffered from the header mismatch problem (14%), 11 files suffered from the failure of header discovery problem (2%) and 5 files didn't have the headers supplied by the user (1%). The set of input files was automatically partitioned into the correct and incorrect categories and the latter was further partitioned into the three sub-categories. The system took a total of 15 minutes to complete the extraction over this data set. A sample the output generated by our system (loaded in Microsoft Excel) on the table in Figure 1 is shown in Figure 4. The system provides a facility to generate different output formats from an intermediate representation of the table.

## 4 Related Work

The importance of table extraction from documents has been recognized in [2]. The work in [5] proposes a text block detection methodology that depends not only on the spatial layout of documents but its linguistic content as well. The text block merging algorithm in that work is based on setting parameters such as the minimum distance between blocks to merge. It is mentioned that such settings can be a source of errors during column extraction. Our approach is essentially syntactic. We use a single user-specified parameter, the minimum column gap, which can be adjusted based on automatically

identified incorrect extractions and the system rerun on them to get higher yield. The work of [7] is based on feature extraction and adaptation of learning techniques for automatically recognizing table boundaries and its rows and columns. The algorithm requires training from a set of sample documents and yields a specialized table extractor with high accuracy on similar documents. Firstly this approach does not yield a generic table extractor such as ours. Secondly, since columns are extracted by detecting certain vertical lines as column separators, it would fail to extract correctly when columns overlap. In contrast our clustering technique can handle overlapping columns. The extraction framework of [6], depends on manual construction of a domain dependent lattice of regular expressions. The algorithm depends on various empirical thresholds which are not easy to discover. Moreover the approach does not appear robust especially when columns can have elements missing in some of its rows. Further it is not clear how regular expressions can handle column elements that span multiple words and data values of different types. Work that can be most closely related to our system are Nodose and Tintin. Nodose is an interactive system for semi-automated data extraction from documents [1]. It utilizes features like beginning and ending marker keywords or fixed offsets to generate extraction wrappers. As far as table extraction is concerned it can only deal with tables which have fixed column separators. Tintin is a system for retrieving text tables [8]. Its main focus appears to be querying and indexing of the extracted data. In contrast we assume that the extracted data will be used to populate a traditional database system which can be queried using standard database techniques. As far as table detection and column extraction is concerned, Tintin uses heuristics to extract both the table and its column data. Specifically for column extraction it tries to discover a gap which runs across all the rows of the table. This approach suffers from two drawbacks. Firstly it can erroneously split a column consisting of multi-word items into two columns. Secondly it is brittle in the sense that it will not be able to detect column boundaries when such a gap does not exist due to misalignments. While we do not address the question of table detection as is done in the Tintin system, we discuss in the next section how clustering ideas can be used for developing robust table detection algorithms also. Clustering algorithms have been extensively investigated in machine learning, image processing and data mining disciplines. While clustering algorithms in these areas can be broadly viewed as an optimization technique, our algorithm is geared towards doing a correct extraction of table which is a novel application of clustering. It will be interesting to investigate the connections between optimizing the measures we have used and correct extraction.

## 5 Concluding Remarks

We have proposed a clustering technique for extracting tabular data from text tables. We have obtained a extraction yield of over 80% on over 500 tables containing financial and parts data on the very first run. We were able to increase the yield to over 90% by readjusting the configuration parameter, i.e. the estimated column gap and re-running the extraction process on those tables which in the first run exhibited header mismatch because of underestimating of the gap value.

On the implementation side it will be useful to integrate an automated table detection algorithm into our system. It is possible to extend the clustering ideas described in this paper to detect tables embedded in arbitrary text. The main idea is to define a closeness metric between rows using the spatial location of gaps in the rows. Based on this metric and the observation that rows in a table are clustered together, we can use our iterative clustering algorithm to detect tables also.

Finally, note that we have used a *static* measure of gaps between clusters. It would be interesting to investigate if we can identify a *dynamic* measure of gaps and merge clusters based upon that. Such a measure might yield correct extractions over a larger class of tables.

## References

- [1] Brad Adelberg. Nodose - a tool for semi-automatically extracting semi-structured data from text documents. In *ACM SIGMOD Conference on Management of Data*, pages 283–294, 1998.
- [2] Douglas Appelt and David Israel. Tutorial notes on building information extraction systems. In *Fifth Conference on Applied Natural Language Processing*, 1997.
- [3] Daniel Fasulo. An analysis of recent work on clustering algorithms. 1999.
- [4] Jeffrey Friedl. *Mastering Regular Expressions*. O Reilly, 1997.
- [5] Matthew Hurst. Layout and language: An efficient algorithm for detecting text blocks based on spatial and linguistic evidence. In *Document Recognition and Retrieval VIII*, 2001.
- [6] Stephen W. Liddle, Douglas M. Campbell, and Chad Crawford. Automatically extracting structure and data from business reports. In *Proceedings of the Eighth International Conference on Knowledge Management*, pages 86–93, 1999.
- [7] H. Ng, C. Lim, and J. Koo. Learning to recognize tables in free text. In *Proceedings of the 37th Annual Meeting of ACL*, pp. 443-450., 1999.
- [8] Pallavi Pyreddy and W. Bruce Croft. Tintin: A system for retrieval in text tables. In *ACM Digital Libraries Conference*, 1997.
- [9] Ian H. Witten and Eibe Frank. *Data Mining*. Morgan Kaufmann Publishers, 1999.