Mining for Attributes and Values in Tables

Nattapon Harnsamut, Naiyana Sahavechaphan
Large-scale Simulation Research Laboratory
National Electronics and Computer Technology Center
Pathumthani, Thailand
nattapon.harnsamut|naiyana.sahavechaphan@nectec.or.th

ABSTRACT: *Table has been recognized as a simply and widely used data representation scheme. Each table alone typically contains rich and useful information which is valuable for many applications such as information retrieval, question-answering and etc. While all table formats can simply be parsed by human, this parsing is difficult for computer, prohibiting such applications to be done in an automatic manner. In this paper, we thus propose the comprehensive and novel table interpretation technique, namely* `tInterpreter`*. Essentially, it transforms a table into its corresponding horizontal 1-dimensional tables. To achieve this, the underlying work is based on (i) the similarity of two given cells with respect to the data type and the semantic correspondence concerns; (ii) the discovery for the boundary of a primitive table residing in a composite table; (iii) the identification of the attribute-value relationship and the value association of cells; and (iv) the integration of two pieces of similar or dissimilar information. The experimental result showed that the overall effectiveness of* `tInterpreter` *was higher than* `Chen`, `Tengli` *and* `Kim`*.*

Keywords: Value mining, Data mining, Data tables, Data relationship

Received: 11 July 2010, Revised 1 August 2010, Accepted 7 August 2010

1. Introduction

Table has been recognized as a simply and widely used data representation scheme [1,2,3]. Its main purpose is to serve human being in understanding data. In particular, it consists of a set of cells wherein each individual cell is defined as either an *attribute* or a *value* cell. These cells can be sat or arranged in any fashions that is appropriate for presenting the underlying data. It is thus likely that a value cell can be defined based on one or more attribute cell(s). In addition, two or more particular value cells are associated and hence must be interpreted altogether. Consider Table 1 as an example. Here, the cell `stnCode` is an attribute of the value cell `PN`. The three value cells 1:00, 1 and 2.8 are associated.

Each table alone typically contains rich and useful information which is valuable for many applications such as information retrieval, question-answering and etc. However, tables can be in different shapes or formats, ranging from simple to complex ones. The format of a table in particular can be given with respect to 4 aspects: *dimension, spanning, corresponding* and *combining* astects. These aspects altogether result in a diversity of table formats. While all table formats can simply be parsed by human, this parsing is difficult for computer, prohibiting the above applications to be done in an automatic manner. There is thus a need for the *table interpretation* technique [1,4] that determines the attribute-value relationship as well as the value association of cells in a table.

Several approaches [2,5,6,7,8] with respect to table interpretation techniques have been proposed in the literature. Wang et al. [8] and Embley et al. [6] rely on the simple heuristic that considers cells in the first row and/or column as attributes. Chen et al. [5] use heuristic rules and similarity of row and column, that are based on the string, named entity and number category, to determine how a table should be interpreted. The table boundary detection along with the round-based analysis are used to deal with a composite table [9]. Tengli et al. [2] use the vector space model to differentiate the label cells (attribute) and the data cells (Value) and apply heuristic rule to manage super rows. In particular, the row (column) whose cells firstly correspond to the labels learnt by vector space model is considered as attribute row (column), while the rest presents the values. Kim et al. [7,10] use visual and semantic coherency checkup to evaluate how to interpret a table. The visual coherency checkup is done based on the number of cells having the major formatting feature and data type, the semantic coherency checks for the semantic correspondence between an attribute and its value.

stnCode			PN	
stnName			Pakpanang	
Time/Date	1	2	3	4
AM.				
1:00	2.83	2.81	2.80	2.71
2:00	2.91	2.86	2.81	2.68
3:00	2.97	2.96	2.90	2.70
PM.				
1:00	3.20	3.03	3.11	3.03
2:00	3.22	3.01	3.06	2.98
3:00	3.22	3.00	3.02	2.91

Table 1. The Tidal Table ($[v, sp] \times_v [hv, sr]$)

While these approaches take step into the right direction, each approach is limited in the *quality* of the information retrieved, resulting in too many (often irrelevant) or in some cases too few or no information. Chen, for example, interpret 2-dimensional tables via both row- and col-wise manners. However, this interpretation is independently done, there by disconnecting the value association. Tengli always interprets the composite tables in a similar way to the 1-dimensional tables. This thus annoys the relationship and association of cells. Kim has not taken into account the significance of super rows and the mutual relationship of information across tables embedded in the composite tables. This thus raises problems in a similar manner to Tengli.

To address these drawbacks, in this paper, we thus propose the comprehensive and novel table interpretation technique, namely `tInterpreter`. Specifically, it is based on the transformation of a table regardless of formats into its corresponding horizontal 1-dimensional table. This transformation is accomplished via the four essential components: (i) *preprocessor* - that simplifies spanning cells and super rows into their appropriate forms to facilitate the process of the rest components; (ii) *boundary discovery* - that applies the similarity of data types in order to find the boundary of each intrinsic primitive tables residing in the composite table; (iii) *analyzer* - that utilizes the semantic correspondence and hence analyzes the attribute-value relationship and the value association of cells in a boundary; and finally (iv) *integrator* - that integrates different pieces of the analyzed information and then gradually forms the desirable horizontal 1-dimensional table. The experimental results has showed that our tInterpreter potentially supports the applications such as information retrieval and question answering much better than other approaches: Chen, Tengli and Kim.

Roadmap. The rest of this paper is organized as followed: Section 2 and 3 describes the details of cells as well as table formats. Our `tInterpreter` is presented in Section 4. Experimental evaluation is given in Section 5. Section 6 concludes the paper.

2. Attribute and Value

A table in general encapsulates a set of cells. Each individual cell can be defined as either an *attribute* or a *value* cell. Specifically, an *attribute* cell, labeled with *generic* and *semantic* term(s), represents a meta-data that describes a particular set of values. Examples of attribute labels and their corresponding value labels are illustrated in Table 2. A *value* cell, on the other hand, stands for a data content or simply an instance of an attribute cell. Its label is typically constrained by a data type and pattern. Examples of types and their corresponding patterns taken from [10] are shown in Table 3. Clearly, a cell is strongly determined as a *value* cell once its label corresponds to any data patterns in Table 3. A cell with no pattern or free-text, however, is either an *attribute* or a *value* cell depending on its purpose as per a given table. It should be noted that a pattern-less value cell has `String` data type and is typically labeled with either specifically meaningful or meaningless term(s) when comparing to its attribute cell. For example, `DP9LAX01AB` and `United Airline` are values for `Tour Code` and `Airline` attributes, respectively.

3. Table Formats

Tables can be in different shapes or formats that are most applicable for their underlying information. Ideally, a table can be considered as a *primitive* or a *composite* table. A primitive table is a table that encapsulates the attribute cells along with their corresponding value cells, and cannot be broken down into any tables. A composite table [9], on the other hand, is a table that composes of two or more primitive tables.

Attributes	Values
Month	January, February, ..., December
Airline	United Airline
Tour Code	DP9LAX01AB
Valid Date	1999-04-01 - 2000-03-31

Table 2. Examples of Attribute and Value Labels.

Types	Patterns
Day	[1|2|...|31]
Month	[[[1|2|...|12] | [*Jan*|*Feb*|...|*Dec*] | ...]
Date	[*dddd-dd-dd* | *dd-dd-dd* | ...]
Percentage	[*d*% | *d.d*% | ...]
Height	[d^+ | $d^+.d^+$]
Weight	[d^+ | $d^+.d^+$]
e-mail	[*a-zA-Z*]$^+$ @[*a-z*]$^+$.[*a-z*]$^+$
Company	[*a-zA-Z*]$^+$ Co. Ltd

Table 3. Examples of Data Types and Patterns

The format of a primitive table can be given with respect to two essential aspects: *dimension*, and *spanning* aspects. Each of which has its own way to present and hence interpret the underlying information.

Dimension Aspect. A table can be created based on either 1-or 2-dimensional fashion. In a 1-dimensional table (or 1-D table), attribute as well as value cells are typically sat or arranged in a horizontal or vertical manner. Here, each attribute cell would describe its underlying value cells via a column-wise or row-wise style, respectively. In addition, there is an association of value cells sitting in the same row or column. In a2-dimensional-table (or 2-D table), value cells are typically presented in a perpendicular manner. There is thus an association of every appropriate three perpendicular value cells. Their attribute cells, on the other hand, are sat on the first row or at the upper left cell (ULC) [3]. We term *h, v* and *hv* as a horizontal 1-D table, a vertical 1-D table and a 2-D table, respectively.

Spanning Aspect. A table can be shaped by one or more spanning cell(s) and/or super row(s). A spanning cell spans two or more cells in a row-wise or column-wise manner. A super row, on the other hand, consumes all cells in a particular row and acts in a similar manner to a row-wise spanning cell. Particularly, each of which aims for *hierarchically grouping* its underlying information or *corresponding* to the overall number of cells in a row-or column-wise direction. It may represent either an attribute or a value. As per an attribute, it is clearly used for describing its underlying information. Otherwise, it must be mutually interpreted with its underlying information. We term *sp* and *sr* as spanning cell and supper row, respectively.

Similarly, the format of a composite table is defined based on two aspects: *corresponding* and *combining* aspects.

Corresponding Aspect. A composite table may consist of *similar* or *dissimilar* primitive tables to accommodate human in *comparing, summarizing* and *understanding* their information altogether. Specifically, two primitive tables are said to be similar when they have the same attributes. Here, they thus contain homogeneous information that should be evenly treated and hence interpreted. Conversely, two primitive tables are dissimilar when different sets of attributes are defined. Here, they thus present heterogeneous but related information that should be mutually interpreted.

Combining Aspect. The primitive tables may be combined in either a *horizontal* or a *vertical* manner to form a composite table. Specifically, these two manners are applicable for combining the similar primitive tables, while the dissimilar primitive tables are typically combined in a vertical style.

Specifically, the combination of these aspects can result in tables in various formats, ranging from simple to complex ones. For simplicity, in the following sections, we use the notation of [*d, s*] to define the format of a particular primitive table where *d* and *s* denote terms specified in the dimension and spanning aspects respectively. The notation [*d, s*] (*op* [*d,s*])$^+$ is to describe the format of a composite table consisting of *n* primitive tables. The *op* includes $+_h$,$+_v$, $\times_h$ and $\times_v$ where $+_h$ and $+_v$ stand for the horizontal and vertical combination of similar primitive tables and the rest are for dissimilar primitive tables. Examples of table formats are given in Table 4. Here, Table 4 (a) - 4(i) illustrates the table format diversity for presenting tidal information hourly observed at Pakpranang station in January, 2008. Table 4(m) - 4(p) are tables samples with 2 columns while Table 4(n) - 4(o) are for 2 rows. Finally, Table 4(q) illustrates a table with *String* and non-*String* values.

Time	Day	Height
0:00	1	2.73
1:00	1	2.83
2:00	1	2.91

(a) [*h*, −]

Time	0:00	1:00	2:00
Day	1	1	1
Height	2.73	2.83	2.9

(b) [*v*, −]

Time/Day	1	2
0:00	2.73	2.82
1:00	2.83	2.81
2:00	2.91	2.86

(c) [*hv*, −]

Time	Day	
	1	2
0:00	2.73	2.82
1:00	2.83	2.81
2:00	2.91	2.86

(d) [*hv, sp*]

PN Station		
Time	Day	Height
0:00	1	2.73
1:00	1	2.83
2:00	1	2.91
3:00	1	2.97
4:00	1	3.05

(e) [*h, sp*]

Time	Day	Height
AM.		
1:00	1	2.83
2:00	1	2.91
PM.		
1:00	1	3.20
2:00	1	3.22

(f) [*h, sr*]

Time	Day	Height	Time	Day	Height
0:00	1	2.73	0:00	2	2.82
1:00	1	2.83	1:00	2	2.81
2:00	1	2.91	2:00	2	2.86
3:00	1	2.97	3:00	2	2.96
4:00	1	3.05	4:00	2	3.01

(g) [*h*, −] $+_h$ [*h*, −]

stnCode	Year	Month
PN	2008	1
Time	Day	Height
0:00	1	2.73
1:00	1	2.83
2:00	1	2.91

(h) [*h*, −] $\times_v$ [*h*, −]

Time	Day	Height	Time	Day	Height	Time	Day	Height
0:00	1	2.73	0:00	2	2.82	0:00	3	2.83
1:00	1	2.83	1:00	2	2.81	1:00	3	2.83
2:00	1	2.91	2:00	2	2.86	2:00	3	2.73

(i) [*h*, −] $+_h$ [*h*, −] $+_h$ [*h*, −]

Time	Day	Height
0:00	1	2.73
1:00	1	2.83
2:00	1	2.91
Time	Day	Height
0:00	2	2.82
1:00	2	2.81
2:00	2	2.86

(j) [*h*, −] $+_v$ [*h*, −]

Time	0:00	1:00	2:00	Time	0:00	1:00	2:00
Day	1	1	1	Day	2	2	2
Height	2.73	2.83	2.91	Height	2.82	2.81	2.86

(k) [*v*, −] $+_h$ [*v*, −]

Time	0:00	1:00	2:00
Day	1	1	1
Height	2.73	2.83	2.91
Time	0:00	1:00	2:00
Day	2	2	2
Height	2.82	2.81	2.86

(l) [*v*, −] $+_v$ [*v*, −]

Station	MaxHeight
PN	3.23
SC	3.45
SA	4.01
PT	3.30

(m) [*h*, −]5×2

Station	PN	SC	SA	PT
MaxHeight	3.23	3.45	4.01	3.30

(n) [*v*, −]2×5

Temperature	Humidity	Precip. today	Wind	Barometer
66◦	75%	n/a	SE at 13 mph	30.02 in

(o) [*h*, −]2×5

Temperature	66◦
Humidity	75%
Precip. today	n/a
Wind	SE at 13 mph
Barometer	30.02 in

(p) [*v*, −]5×2

DATE	OPPONENT	TIME	SCORE
1	Washington	7:30	4-2
2	Ottawa	7:30	7-3
6	Edmonton	7:30	5-2
8	TampaBay	7:30	5-5

(q) [*h*, −]*String*

Table 4. Table Formats

4. Table Interpreter

The diversity of table formats hinder applications such as information retrieval and question answering. It is thus necessary to transform all table formats into a unified format. Essentially, such unified format must be able to (i) preserve all information contained in an original format; and (ii) support each particular information query ranging from a simple to complex one. For example, as per Table 4(a), the two queries can be issued for the `Height` value: (i) `Time` = 1:00 and (ii) `Time` = 1:00 and `Day` = 1. In this work, we have chosen a horizontal 1-D table format as a unified format because it is not only simple but also meets the above two qualifications. We thus propose a table interpretation technique, namely `tInterpreter`, based on the transformation of a given table into its corresponding horizontal 1-D table format. In particular, `tInterpreter` takes a table $\mathcal{T}$ (a source table) an input and produces as output its corresponding horizontal 1-D table $\mathcal{T}'$ (a target table).Figure 1 gives an architectural overview of `tInterpreter`, and highlights its four key components: *Preprocessor, Boundary Discovery, Analyzer* and *Integrator*. Their details are given in the following sections.

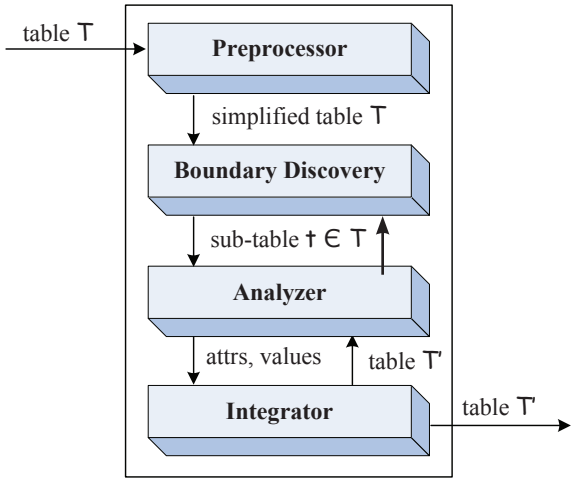

Figure 1. The `tInterpreter` Architecture

4.1 Preprocessor

The existence of spanning cells and super rows complicates the overall process of table interpretation. There is thus a need for simplifying each individual table shaped with the spanning cells and super rows. In particular, a table should no longer contain the spanning cells and super rows. Their representations, however, must be preserved in another appropriate form such that their purpose would be later achieved. This table simplification is accomplished by the *Preprocessor* component. Specifically, each spanning cell is split into two or more cells depending on the number of cells it has originally spanned. The first cell has its label and the rest is filled with the symbol #. The super rows, on the other hand, are replaced by a newly created, left, column. It is filled with the labels of super rows in a similar manner to the row-wise spanning cells. Consider Table 1 as an example. Here, it has four spanning cells `StnCode`, `PN`, `StnName` and `Pakpanang` and two super rows `AM`. and `PM`.. Accordingly, it is simplified as shown in Table 5.

	stnCode	#	#	PN	#
	stnName	#	#	Pakpanang	#
	Time/Day	1	2	3	4
AM.	1:00	2.83	2.81	2.80	2.71
#	2:00	2.91	2.86	2.81	2.68
#	3:00	2.97	2.96	2.90	2.70
PM.	1:00	3.20	3.03	3.11	3.03
#	2:00	3.22	3.01	3.06	2.98
#	3:00	3.22	3.00	3.02	2.91

Table 5. The Simplified Version of Tidal Table 1

4.2 Boundary Discovery

Basically, three steps are needed when comprehending a source table $\mathcal{T}$: (i) *identifying* the boundary $\mathcal{B}$ of each primitive table in $\mathcal{T}$; (ii) *analyzing* the attribute value relationship as well as the value association among cells of each primitive table in $\mathcal{T}$; and (iii) *integrating* different pieces of analyzed information alto gether to form a target table $\mathcal{T}'$. The identification task is accomplished by the *BoundaryDiscovery* component, while the rest are done by the *Analyzer* and *Integrator* components (details in Section 4.3 and 4.4). Specifically, in the *BoundaryDiscovery* component, the boundary $\mathcal{B}$ is determined based on (i) the similarity of two consecutive rows starting from the bottom-most row (or two consecutive columns starting from the right-most column); and (ii) the discovery for a column (or row) whose most cell shave the symbol # or none. We believe that the data pattern alone can be sufficient for deciding whether two given rows (or columns) are similar. The similarity of two rows t_i and t_j, given by $rSim(t_i, t_j)$, is thus formally defined as per Equation 1. Similarly, Equation 2 defines the similarity of two columns t_k and t_l, given by $cSim(t_k, t_l)$.

$$rSim(\mathbf{t_i}, \mathbf{t_j}) = \frac{\sum_{y=k}^{l} PtnMatch(Ptn(t_{iy}), Ptn(t_{jy}))}{l-k+1} \tag{1}$$

$$cSim(\mathbf{t_k}, \mathbf{t_l}) = \frac{\sum_{x=i}^{j} PtnMatch(Ptn(t_{xk}), Ptn(t_{xl}))}{j-i+1} \tag{2}$$

where, as per a table $\mathcal{T}$ or $\mathcal{T}_{m\times n}$ where m (n) is the number of rows (columns), t_i (t_j) represents the i^{th} (j^{th}) row of a table $\mathcal{T}$ ($1 \le i, j \le m$ and $i \ne j$), t_k (t_l) the k^{th} (l^{th}) column of a table $\mathcal{T}$ ($1 \le k, l \le n$ and $k \ne l$), t_{xy} the cell of a table $\mathcal{T}$ sitting on the x^{th} row and y^{th} column ($1 \le x \le m$ and $1 \le y \le n$), *Ptn* the function that determines the data pattern (see Table 3) of a specified cell and *PtnMatch* the function that determines the exact match of two given data patterns. It should be noted that the pattern of a cell does not correspond to what given in Table 3 can be either the symbol # or free-text. The symbol # has no data type, while the `String` data type depicts the free-text.

Essentially, the underlying process of the *BoundaryDiscovery* component is shown via the pseudocode given in Figure 2. Here, as per a table $\mathcal{T}_{m\times n}$, the process starts with its lower right cell (LRC) or simply an origin of (m, n). The similarity of two consecutive rows (columns) along with the discovery for the symbol # in a column-wise (row-wise) direction is then determined to find the first boundary $\mathcal{B}$ of a table $\mathcal{T}$. Next, as per a boundary $\mathcal{B}$, it works side-by-side with the *Analyzer* and *Integrator* components. It later checks for new origins to ensure that all remaining cells of a table $\mathcal{T}$ are performed. For each new origin, it is recursively proceeded. Finally, it stops when there is no more origin.

4.3 Analyzer

Once the boundary $\mathcal{B}$ of a table is determined, the next step is to analyze the attribute-value relationship and the value association among cells within a boundary $\mathcal{B}$. This task is accomplished by the *Analyzer* component. Specifically, its underlying work is based on (i) the size and the row (column) similarity of aboundary $\mathcal{B}$ given by the *BoundaryDiscovery* component; (ii) the upper row of aboundary $\mathcal{B}$. In particular, when the label defined in its upper left cell (ULC) contains the symbol "/" oris empty, there is an association of three perpendicular value cells in $\mathcal{B}$. Two out of these value cells are described by the ULC as their attribute cell. Similar analysis can also be performed when most data patterns of cells in such upper row correspond to those in Table 3; (iii) the symmetry of attribute cells in an upper row (or left column) of aboundary $\mathcal{B}$. Essentially, the attribute symmetry represents the horizontal or vertical combination of similar primitive tables (see Table 4(j) and 4(i) as examples). The indexes of the vector $\mathcal{V}$ of attribute cells in a row or column that begins all primitive tables, given by Symmetry($\mathcal{V}$), is formally defined as per Equation 3. (iv) the attribute-value relationship of two given cells. Essentially, a cell t_{xy} is an attribute of a cell $t_{x'y'}$ (t_{xy} describe $t_{x'y'}$) only if the semantic of a label defined in t_{xy} is either (a) similar to it of a data type of $t_{x'y'}$ (see Table 3) or (b) describes it of a label in $t_{x'y'}$. Formally, the attribute-value relationship degree of two cells t_{xy} and $t_{x'y'}$, given by $AttrVal(t_{xy}, t_{x'y'})$, is defined as per Equation 4 and 5 respectively. The Equation 4 would be performed when a cell $t_{x'y'}$ has data type (except `String`) corresponding to any in Table 3. Otherwise, Equation 5 would be performed; and (iv) the attribute-value relationship of a cell $t_{xy} \in \mathcal{T}$ as an attribute and a column $c' \in \mathcal{T}'$ as a set of values. Basically, its computation is based on the attribute-value relationship of two cells. Equation 6, given by \$Column\$ (t_{xy}, $\mathcal{T}'$), formally defines the selection of a column $c' \in \mathcal{T}'$ that most satisfies an attribute t_{xy}.

$$Symmetry(\mathcal{V}) = \{i, j | \forall (Label(v_{i+k}) = Label(v_{j+k}))\} \tag{3}$$

BoundaryDiscovery(Table $\mathcal{T}$, Origin (row x_2, col y_2)) {
1. if ($x_2 = y_2 = 2$) or ($x_2 = 2$ and $y_2 \geq 3$) or ($x_2 \geq 3$ and $y_2 = 2$)
   row $x_1 \leftarrow 1$, column $y_1 \leftarrow 1$
   go to step 5
2. find the first row x_1 of a boundary $\mathcal{B}$ by
   comparing the similarity of two consecutive rows
   (start from the bottom-most row x_2)
   until the similarity is below threshold or no more row
   row $x_1 \leftarrow$ the latest row similar to its lower row
3. if at least two rows are similar
   scan columns until the # is found or no more column
   col $y_1 \leftarrow$ the latest column without #
   go to step 5
4. find the first column y_1 of a boundary $\mathcal{B}$ by
   comparing the similarity of two consecutive columns
   (start from the right-most column end_{col}) until
   the similarity is below threshold or no more column
   col $y_1 \leftarrow$ the latest column similar to its right column
   scan rows until the # is found or no more row
   row $x_1 \leftarrow$ the latest row without #
5. boundary $\mathcal{B} \leftarrow (x_1, y_1)$ to (x_2, y_2)
   where $1 \leq x_1 \leq x_2 \leq m$, $1 \leq y_1 \leq y_2 \leq n$
6. call the *Analyzer* component to analyze cells
   within a boundary $\mathcal{B}$ (Section 4.3)
   analyzed information $\mathcal{I} \leftarrow$ *Analyzer*($\mathcal{T}$, $\mathcal{B}$)
   call the *Integrator* component to integrate the analyzed
   information into a resulting table $\mathcal{T}'$ (Section 4.4)
   table $\mathcal{T}' \leftarrow$ *Integrator*($\mathcal{T}'$, $\mathcal{I}$)
7. determines new origins from the rest parts of a table $\mathcal{T}$
   origin $\mathcal{O} \leftarrow \{(x_2, y_1\text{-}1, ...), (x_1\text{-}1, y_2, ...), ...)\}$
   for each $o \in \mathcal{O}$
   recursively call *BoundaryDiscovery*($\mathcal{T}$, o)
}

Figure 2. The *BoundaryDiscovery* Pseudocode

where $\mathcal{V}$ is the vector of attribute cells in a row or column, an attribute cell $v \in \mathcal{V}$, \$Label\$ the function the returns the label of a specified cell v, $1 \leq i \leq n$, $j = i + rn$, $0 \leq k \leq n-1$, $1 \leq r \leq \frac{|\mathcal{V}|}{n}$, and n the number of attribute cells in all similar primitive tables. Consider Table 4(g) and 4(i) as examples. Here, in Table 4(g), *Symmetry*($\mathcal{V}$) is {1,4} when $k = 3$. *Symmetry*($\mathcal{V}$) is {1,4,7} for Table 4(i) and $k = 3$.

$$AttrVal(t_{xy}, t_{x'y'}) = max_{\alpha \in Type(t_{x'y'})} lSim(Label(t_{xy}), \alpha) \tag{4}$$

$$AttrVal(t_{xy}, t_{x'y'}) = lSim(t_{xy}, t_{x'y'}) \tag{5}$$

where t_{xy} ($t_{x'y'}$)the cell of a table $\mathcal{T}$ sitting on the x^{th} (x'^{th}) row and the y^{th} (y'^{th}) column, *Type* the function that determines the data type with respect to the data pattern of a specified cell, and *lSim* the function that determines the semantic correspondence of two given labels. Basically, *lSim* is computed based on WordNet [11] and formula given in [12].

$$Column(t_{xy}, \mathcal{T}') = argmax_{j=1,..,n}\{\frac{\sum_{i=1}^{m} AttVal(t_{xy}, t'_{ij})}{m}\} \tag{6}$$

where t'_{ij} is a cell in a target table $\mathcal{T}'$ sitting on the i^{th} row and j^{th} column, m the number of row in $\mathcal{T}'$ and n the number of column in $\mathcal{T}'$.

The underlying process of the *Analyzer* is shown via the pseudocode given in Figure 3. Here, as per a boundary $\mathcal{B}$, the process firstly determines the boundary property along with the underlying seven analysis cases. Once the

desirable analysis case has been chosen, its underlying work would thus be performed. Consider Table 4(c) and 4(g) as examples. In Table 4(c) [*hv*,−], the row- and column-wise analysis as described in case 4 is applied, resulting in *attr* :<`time,day,->` and *val* :< 0 : 00,1,2.73 >,< 0 : 00,2,2.82 >,...,< 2 : 00,2,2.86 >. The column-wise analysis is done for Table 4(g). This returns `attr:<time,day, height>` and *val* :< 0 : 00,1,2.73 >,...,< 4 : 00,1,3.05 >,< 0 : 00,2,2.82 > , ...,< 4 : 00,2,3.01>.

```
Analyzer(table 𝒯, boundary ℬ) {
  // recall that ℬ covers area from (x1,y1) to (x2,y2)
  // recall that t_xy ∈ ℬ ⊂ 𝒯 where x1 ≤ x ≤ x2, y1 ≤ y ≤ y2
  the number of matched rows p ← x2 - x1 + 1;
  the number of matched columns q ← y2 - y1 + 1;
  case 1: ℬ2×2
    if (t_x1y1 describes t_x2y1) and (t_x1y2 describes t_x2y2)
      analyze a boundary ℬ via a column-wise manner
        ℐ ← attr:<t_x1y1, t_x1y2> and val:<t_x2y1, t_x2y2>
    else analyze a boundary ℬ via a row-wise manner
        ℐ ← attr:<t_x1y1,t_x2y1> and val :<t_x1y2,t_x2y2>
  case 2: ℬ2×q (q > 2)
    if columns are similar
      analyze a boundary ℬ via a row-wise manner
      ℐ ← attr :<t_x1y1, t_x2y1> and val :<t_x1k, t_x2k > where y1+1≤k≤y2
    else analyze a boundary ℬ via a column-wise manner
      ℐ ← attr :<t_x1y1,...,t_x1y2> and val :<t_x2y1,...,t_x2y2>
  case 3: ℬp×2 (p > 2)
    if rows are similar
      analyze a boundary ℬ via a column-wise manner
      ℐ ← attr :<t_x1y1, t_x1y2> and val :<t_ky1, t_ky2> where x1+1≤k≤x2
    else analyze a boundary ℬ via a row-wise manner
      ℐ ← attr :<t_x1y1,...,t_x2y1> and val :<t_x1y2,...,t_x2y2>
  case 4: ℬp×q (p, q > 2) whose rows are similar
    𝒱 ← vector of cells in the (x1 − 1)^th row of 𝒯
    if (the left-most cell v_y1 ∈ 𝒱 contains "/" )
      analyze ℬ with 𝒱 via a row- & column-wise manners
      ℒ1 ← 1^st label of v_y1 and ℒ2 ← 2^nd label of v_y1
      if ℒ1 describes t_xy1 where x1 ≤ x ≤ x2
        ℐ ← attr :<ℒ1, ℒ2,null> val: <t_xy1, v_k, t_xk> where x1≤x≤x2, y1+1≤k≤y2
      else ℐ ← attr :<ℒ2, ℒ1,null> val: <t_xy1, v_k, t_xk> where x1≤x≤x2, y1+1≤k≤y2
    else if (the left-most cell v_y1 ∈ 𝒱 is empty)
      ℐ ← attr :<null,null,null> val: <t_xy1, v_k, t_xk> where x1≤x≤x2, y1+1≤k≤y2
    else if most data patterns of cells v_y1+1,...,v_y2 ∈ 𝒱 corresponds to Table 3
      ℐ ← attr :<v_y1,null,null> val: <t_xy1, v_k, t_xk> where x1≤x≤x2, y1+1≤k≤y2
    else if —𝒮 ← Symmetry(𝒱)— > 0 // symmetry
      analyze ℬ via a column-wise and symmetric manner
      i ← the smallest index in 𝒮
      n ← the difference of every two indexes in 𝒮
      ℐ ← attr :<v_i,...,v_i+n−1 > val :<t_xs,..,t_xs+n−1 > for all s ∈ 𝒮 and x1 ≤ x ≤ x2
    else analyze ℬ with 𝒱 via a column-wise manner
      ℐ ← attr :<v_y1,..,v_y2> where v ∈ 𝒱 val :<t_xy1,..,t_xy2> where x1≤x≤x2
    x1 ← x1 − 1
  case 5: ℬp×q (p, q > 2) whose columns are similar
    done in an opposite manner to case 4
  case 6: ℬp×1 (p >1) having #
    𝒱 ← vector of cells in the (x1-1)^th row of 𝒯
    if v_y1 ∈ 𝒴 describes any column c ∈ 𝒯'
      ℐ ← attr :<v_y1> col_𝒯' :<c >
    else ℐ ← val :<v_y1> row_𝒯 :<x1-1,...,x2>
    y1 ← y1 − 1
  case 7: ℬ1×q (q >1) having #
    𝒱 ← all cells t_xy ∈ ℬ filled with any labels except #
    if (there are only two cells v1 and v2 ∈ 𝒴) and
      (v1 describes v2)
        ℐ ← attr :<v1> and val :<v2>
    else if all v ∈ 𝒱 describes columns c ∈ 𝒯'
      ℐ ← attr :<all v ∈ 𝒱 > col_𝒯':<all c >
    else ℐ ← val :<all v ∈ 𝒱 >
              col_𝒯:<rang of columns of each v >

  return ℐ
}
```

Figure 3. The *Analyzer* Pseudocode

4.4 Integrator

The last step is to integrate different pieces of analyzed information given by the *Analyzer* component. This task is accomplished by the $Integrator$ component. Specifically, as per an analyzed information $\mathcal{I}$, it accumulatively creates a horizontal 1-D table $\mathcal{T}'$. Its underlying work is based on the intrinsic information encapsulated in $\mathcal{I}$ against the information in a table $\mathcal{T}'$ as shown via the pseu docode given in Figure 4. Essentially, $\mathcal{I}$ is *horizontally* integrated with $\mathcal{T}'$ when $\mathcal{I}$ encapsulates both *attr* and *val* where *attr* is not contained in the current $\mathcal{T}$$. Otherwise, $\mathcal{I}$ is vertically integrated with $\mathcal{T}'$. However, it is likely that $\mathcal{I}$ may contains either *attr* or *val*. With *attr* alone, it would be leveraged as attributes for the specified column of $\mathcal{T}'$. For *val*, it is horizontally integrated with $\mathcal{T}'$ for a certain number of rows in $\mathcal{T}'$.

```
Integrator(table T, table T', boundary B,
           analyzer information I) {
  case 1: I contains attr & val where attr not ∈ T'
    horizontally integrate I with T'
      T' ← I ×_h T'
  case 2: I contains attr & val where attr ∈ T'
    vertically integrate I with T'
      T' ← I ×_v T'
  case 3: I contains attr and col_T'
    attach each a ∈ attr to each cincol_T'
      T' ← T' + attr
  case 4: I contains val and row_T
    R ← locate rows of T' that contains information
          sitting on r ∈ row_T
    horizontally integrate I with T' at row r ∈ R
      T' ← I ×_h T' at row r ∈ R
  case 5: I contains val and col_T
    R ← locate rows of T' that contains information
          sitting on c ∈ col_T
    horizontally integrate I with T' at column r ∈ R
      T' ← I ×_h T' at column r ∈ R
}
```

Figure 4. The *Integrator* Pseudocode

Consider the interpretation of the tidal table $\mathcal{T}_{11\times5}$ as shown in Table 1 as an example. According to our *tInterpreter*, this table is firstly simplified via the $Preprocessor$ component, resulting in a table $\mathcal{T}_{9\times6}$ depicted at Table 5. The simplified table $\mathcal{T}_{9\times6}$ is now performed by the *BoundaryDiscovery* component along with the *Analyzer* and *Integrator* components. Here, there are five rounds going back and forth among these three components.

The first round. *BoundaryDiscovery* starts its process at the origin ($x_2 = 9, y_2 = 6$). It finds the similarity of the 9th and 8th rows and then proceeds until the 4th row. Next, it scans for the symbol # in a column-wise manner and stops at the 2th column. The boundary $\mathcal{B}$ is now ($x_1 = 4, y_1 = 2$) to ($x_2 = 9, y_2 = 6$) as shown in the left table of Table 6 (a). *Analyzer* evaluates the boundary $\mathcal{B}$ and finds that $\mathcal{B}$ meets its 4th analysis case. The analyzed information $\mathcal{I}$ would thus be *attr* :<`time,day,->` and *val* :< `1:00,1,2.83` >, < `1:00,2,2.81` >,....,< `3:00,4,2.91` >. *Integrator* takes this analyzed information $\mathcal{I}$ and forms an initial target table $\mathcal{T}'$ as shown in the right table of Table 6 (b). The new origins would be $\{(x_2 = 9, y_2 = 1),(x_2 = 2, y_2 = 6)\}$.

The second round. With the origin ($x_2 = 9, y_2 = 1$), *BoundaryDiscovery* finds the similarity of two rows, the 9th and 8th. The scan for the symbol # here is not needed as the 1th column is being performed. The boundary $\mathcal{B}$ would thus be ($x_1 = 8, y_1 = 1$) to ($x_2 = 9, y_2 = 1$) (see the left table of Table 6 (b). *Analyzer* determines that the boundary $\mathcal{B}$ corresponds to the 6th analysis case. The analyzed information $\mathcal{I}$ would thus be *val* :<`PM`.> $row_{\mathcal{T}}$:<7,8,9>. Based on this analyzed information $\mathcal{I}$, *Integrator* then performs a horizontal integration between $\mathcal{I}$ and $\mathcal{T}'$ at some appropriate rows of $\mathcal{T}'$. The resulting table $\mathcal{T}'$ is shown in the right table of Table 6 (b). The new origin in this round would be $\{(x_2 = 6, y_2 = 1)\}$. It is combined with the remaining origins, resulting $\{(x_2 = 6, y_2 = 1),(x_2 = 2, y_2 = 6)\}$.

The third round. According to the origin (x_2= 6, y_2=1), *BoundaryDiscovery, Analyzer and Integrator* perform in a similar manner to the second round. The left table of Table 6 (c) illustrates the boundary *B* (x_1=5, y_1=1) to(x_2=6, y_2=1).The resulting table $\mathcal{T}'$ is shown in the right table of Table 6 (c). The new origin in this round would be {(x_2= 3, y_2= 1)}. However, its three upper cells contains none. The overall origin is now {(x_2= 2, y_2= 6)}.

The last round. Based on the origin ($x2$ = 2, y_2 = 6), Bo*undaryDiscovery* comes with the boundary *B* (x_1=1, y_1=1) to(x_2=2, y_2= 6) (see the left table of Table 6 (d)). Next, *Analyzer* component analyzes the boundary B via the 7th case, resulting in the analyzed information $\mathcal{I}$ as {*attr* :<stnName> and *val* :<Pakpanang>, *attr* :<stnCode> and *val* :<PN>}. Finally, *Integrator* has integrated the analyzed information $\mathcal{I}$ with the current table $\mathcal{T}'$ using the 1st case. This results in a table $\mathcal{T}'$ as shown in the right table of Table 6 (d). The process finally finishes as all cells have been evaluated.

	stnCode	#	#	PN	#
	stnName	#	#	Pakpanang	#
	Time/Day	1	2	3	4
AM.	1:00	2.83	2.81	2.80	2.71
#	2:00	2.91	2.86	2.81	2.68
#	3:00	2.97	2.96	2.90	2.70
PM.	1:00	3.20	3.03	3.11	3.03
#	2:00	3.22	3.01	3.06	2.98
#	3:00	3.22	3.00	3.02	2.91

Day	Time	
1	1:00	2.83
1	2:00	2.91
1	3:00	2.97
...	...	....
1	1:00	3.20
1	2:00	3.22
1	3:00	3.22
...	...	...

(a) The First Round

	stnCode	#	#	PN	#
	stnName	#	#	Pakpanang	#
	Time/Day	1	2	3	4
AM.	1:00	2.83	2.81	2.80	2.71
#	2:00	2.91	2.86	2.81	2.68
#	3:00	2.97	2.96	2.90	2.70
PM.	1:00	3.20	3.03	3.11	3.03
#	2:00	3.22	3.01	3.06	2.98
#	3:00	3.22	3.00	3.02	2.91

	Day	Time	
	1	1:00	2.83
	1	2:00	2.91
	1	3:00	2.97
	...	...	....
PM.	1	1:00	3.20
PM.	1	2:00	3.22
PM.	1	3:00	3.22
PM.	...	...	...

(b) The Second Round

	stnCode	#	#	PN	#
	stnName	#	#	Pakpanang	#
	Time/Day	1	2	3	4
AM.	1:00	2.83	2.81	2.80	2.71
#	2:00	2.91	2.86	2.81	2.68
#	3:00	2.97	2.96	2.90	2.70
PM.	1:00	3.20	3.03	3.11	3.03
#	2:00	3.22	3.01	3.06	2.98
#	3:00	3.22	3.00	3.02	2.91

	Day	Time	
AM.	1	1:00	2.83
AM.	1	2:00	2.91
AM.	1	3:00	2.97
AM.	...	...	....
PM.	1	1:00	3.20
PM.	1	2:00	3.22
PM.	1	3:00	3.22
PM.	...	...	...

(b) The Third Round

	stnCode	#	#	PN	#
	stnName	#	#	Pakpanang	#
	Time/Day	1	2	3	4
AM.	1:00	2.83	2.81	2.80	2.71
#	2:00	2.91	2.86	2.81	2.68
#	3:00	2.97	2.96	2.90	2.70
PM.	1:00	3.20	3.03	3.11	3.03
#	2:00	3.22	3.01	3.06	2.98
#	3:00	3.22	3.00	3.02	2.91

stnCode	stnName		Day	Time	
PN	Pakpanang	AM.	1	1:00	2.83
PN	Pakpanang	AM.	1	2:00	2.91
PN	Pakpanang	AM.	1	3:00	2.97
PN	Pakpanang	AM.	...	...	....
PN	Pakpanang	PM.	1	1:00	3.20
PN	Pakpanang	PM.	1	2:00	3.22
PN	Pakpanang	PM.	1	3:00	3.22
PN	Pakpanang	PM.	...	...	...

(b) The Fourth Round

Table 6. Walkthrough the `tInterpreter` Technique per Tidal Table 1

5. Experimental Evaluation

A series of experiments were conducted to evaluate the potential benefits of our `tInterpreter` against other existing approaches. The approaches proposed by Chen [5], Tengli [2] and Kim [7] were chosen for such evaluation.

5.1 Setup

To perform a series of experiments, we firstly developed the `tInterpreter` system on the basis of `tInterpreter` technique. Specifically, the `tInterpreter` evaluated the semantic similarity and relationship of two labels based on the formula given in [12] and WordNet [11]. The repository for the experiments contained data types and patterns defined by [7]. We deployed the `tInterpreter` system along with the repository on a standalone PC Pentium IV 3 GHz with 3 GB of RAM running Ubuntu10.04. Secondly, we designed 26 tables with respect to four different aspects previously defined in Section 3. They are classified into eight categories: *1-D tables*(2×2), *1-D tables*(2×n and n×2), *1-D tables*(m×n), *1-D tables* varying string values, *2-D tables, super-row based tables, composite tables*($+_h$ and $+_v$), and *composite tables* ($\times_h$ and $\times_v$). Essentially, these tables are designed based on formats that are commonly created by any organizations. Here, we focus on the table format diversity rather than analogous table format with large quantity. These tables thus encapsulate RICH formats and hence are sufficiently representative of numerous and real world tables. Lastly, as per a table, we defined its corresponding query set [13]. The query set consisted of one or more queries depending on the number of attributes in that table. All queries in the set asked for a specific value and were varied by conditions. These conditions ranged from a single statement to some numbers of compound statements. Each of which uniquely encapsulated 1,2, ...,or $n-1$ different statements where n was the number of potential attributes. For example, as per Table 4(a), two queries were to ask for the `Height` with respect to:(i) `Time` = 1:00 and (ii) `Time` = 1:00 and `Day` = 1.

Measurement. We use F-measure to evaluate the effectiveness of our `tInterpreter` and other approaches against experts who manually performed the query set as per a given table. In particular, F-measureis defined as a harmonic mean of the average of precision and recall.

5.2 Result

We conducted eight different sets of experiments based on eight categories of experimental tables. For each set, as per its individual table, we initially asked two experts to manually perform all queries q in the query set $\mathcal{Q}$. Similarly, we ran the `tInterpreter` system and manually did walkthrough algorithms proposed by `Chen` [5], `Tengli` [2] and `Kim` [7]). Here, we assumed that the semantic concerns of these approaches were well done. We next computed the average F-measure based on the F-measure given by performing all $q \in \mathcal{Q}$.This average F-measure was then used to compare the effectiveness of these approaches.

5.2.1 Comparison on 1-Dimensional Tables (2 × 2) The first set of experiments was conducted on the two 1-D tables ($[h,-]_{2\times2}$ and $[v,-]_{2\times2}$) whose value patterns corresponded to e-mail and company. Table 7 shows the average F-measure provided by different approaches along with the experimental tables. Our `tInterpreter` and `Kim` performed the best. This is mainly because the semantic similarity significantly facilitated the attribute-value relationship. `Chen` was applicable for a vertical 1-D table alone since a row-wise interpretation was always set as default if neither ”row-wise” nor ”col-wise” can be assigned. `Tengli` poorly performed as attributes and values could not be identified.

Table Formats	`Chen`	`Tengli`	`Kim`	`tInterpreter`
$[h,-]_{2\times2}$	0%	0%	100%	100%
$[v,-]_{2\times2}$	100%	0%	100%	100%

Table 7. Comparison on 1-D Tables (2 × 2)

5.2.2 Comparison on 1-Dimensional Tables (2 × *n* and *n* × 2) The second set of experiments was conducted on the four 1-D tables whose value patterns were time, day and height (non-string data types). The format of these table were $[h,-]_{2\times n}$, $[v,-]_{2\times n}$, $[h,-]_{n\times2}$, and $[v,-]_{n\times2}$. Table 8 shows the average F-measure provided by different approaches along with the experimental tables. Our `tInterpreter` and `Tengli` performed the best due to the table dimension itself along with an application of the data pattern and the vector space model, respectively. `Chen` did not work for the table whose format was $[h,-]_{2\times n}$ because the row-wise interpretation was always accomplished. `Kim` poorly performed because the col-wise interpretation was fixed for table $[v,-]_{2\times n}$ while the row-wise interpretation was for table $[h,-]_{n\times2}$.

Table Formats	Chen	Tengli	Kim	tInterpreter
$[h, -]_{2\times n}$	0%	100%	100%	100%
$[v, -]_{2\times n}$	100%	100%	0%	100%
$[h, -]_{n\times 2}$	100%	100%	0%	100%
$[v, -]_{n\times 2}$	100%	100%	100%	100%

Table 8. Comparison on 1-D Tables ($2 \times n$ and $n \times 2$)

5.2.3 Comparison on 1-Dimensional Tables ($m \times n$) The third set of experiments was conducted on the two 1-D tables whose values patterns were time, day and height (non-`String` data types). The format of these tables were [*h*,−] and [*v*,−]. Here, we found that all approaches equivalently performed with the completion of all queries. This is because these two tables were not only simple but also contained attributes and values corresponding to heuristics of all approaches.

5.2.4 Comparison on 1-D Table with String Values The fourth set of experiments was conducted on the five 1-D tables: $[h,-]_{m\times 5}$. Each of which had different number of columns with `String` data type while the rest columns had non-`String` data type. Table 9 shows the average F-measure provided by different approaches along with the experimental tables. Here, our `tInterpreter`, `Chen` and `Kim` equivalently performed and accomplished only those tables with less number of *String* columns. However, all these approaches were poorly performed. This is mainly due to the fact that the *String* data type violates their heuristics.

Table Formats	Chen	Tengli	Kim	tInterpreter
1 String Column	100%	80%	100%	100%
2 String Columns	100%	67%	100%	100%
3 String Columns	0%	33%	0%	0%
4 String Columns	0%	0%	0%	0%
5 String Columns	0%	0%	0%	0%

Table 9. Comparison on 1-D Table Varying String Pattern Tables

5.2.5 Comparison on 2-Dimensional Tables The fifth set of experiments was conducted on the four 2-D tables. The format of these tables were [*hv*,−] (*mixed*), [*hv*,−], [*hv, sp*] (*mixed*) and [*hv, sp*]. The ”mixed” tables referred to tables that had values with string data type on the first row and column while the rest cells contained values with the non-`String` data types. Unlikely, the ”non-mixed” tables were tables whose all perpendicular values has non-`String` data types. Table 10 shows the average F-measure provided by different approaches along with the experimental tables. Here, our `tInterpreter` and `Kim` performed the best with the completion of all queries. This is mainly because data types significantly facilitated the recognition of 2-D tables. `Chen` and `Tengli` poorly performed, 50% and 25% of F-measure. Particularly, the vector space model and cell similarity proposed by `Tengli` could be applicable for only ”mixed” tables. In addition, `Chen` partially performed as the independent interpretation between the row-and col-wise manners disconnected the value association.

Table Formats	Chen	Tengli	Kim	tInterpreter
[*hv*, −](*mixed*)	50%	100%	100%	100%
[*hv*, −]	0%	0%	100%	100%
[*hv, sp*](*mixed*)	50%	100%	100%	100%
[*hv, sp*]	0%	0%	100%	100%

Table 10. Comparison on 2-D Tables

5.2.6 Comparison on Super-Row Based Tables The sixth set of experiments was conducted on the two super-row based tables: [*h, sr*] and [*hv,sr*] (*mixed*). Table 11 shows the average F-measure provided by different approaches along with the experimental tables. Here, our `tInterpreter` and `Tengli` performed the best as super rows were taken into account as an additional

information to their underlying information. Kim averagely performed due to the fact that super rows and their underlying information was equally treated. Kim could thus complete only queries not involved with the super row concern. Chen performed the worst because super rows affected the process of boundary detection and hence the overall table interpretation.

Table Formats	Chen	Tengli	Kim	tInterpreter
[*h, sr*]	0%	100%	67%	100%
[*hv, sr*](*mixed*)	0%	100%	50%	100%

Table 11. Comparison on Super-Row based Tables

5.2.7 Comparison on Composite Tables whose Primitive Tables are Similar The seventh set of experiment was conducted on the six composite tables whose underlying primitive tables are similar. These table formats were $[h,-]+_h[h,-]$, $[h,-]+_v[h,-]$, $[v,-]+_h[v,-]$, $[v,-]+_v[v,-]$, $[h,-]+_h[h,-]+_h[h,-]$, and $[v,-]+_v[v,-]+_v[v,-]$. In addition, the values of each primitive tables were time, day and height. Table 12 shows the average F-measure provided by different approaches along with the experimental tables. Here, we performed the best as different concerns had been comprehensively taken into account. Chen, Tengli, and Kim did not well for the composite tables that combined primitive tables in a similar manner to the format of primitive tables. This is mainly because information across the embedded primitive tables was not mutually interpreted.

Table Formats	Chen	Tengli	Kim	tInterpreter
$[h, -] +_h [h, -]$	0%	0%	0%	100%
$[h, -] +_v [h, -]$	100%	100%	100%	100%
$[v, -] +_h [v, -]$	100%	100%	100%	100%
$[v, -] +_v [v, -]$	0%	0%	0%	100%
$[h, -] +_h [h, -] +_h [h, -]$	0%	0%	0%	100%
$[v, -] +_v [v, -] +_v [v, -]$	0%	0%	0%	100%

Table 12. Comparison on Composite Tables whose Primitive Tables are Similar

5.2.8 Comparison on Combining Tables whose Primitive Tables are Dissimilar The last set of experiments was conducted on the two composite tables: $[h,-]\times_v [h,-]$, and $[hv, sp]\ (m)\times_v [v,-]$. Table 13 shows the average F-measure provided by different approaches along with the experimental tables. Here, we performed the best as different concerns had been comprehensively taken into account. Kim and Chen poorly performed with 50% and 40% of F-measure. In particular, the value association and the mutual relationship of information tables was neglected. Tengli performed the worst as all composite table were considered as primitive table.

Table Formats	Chen	Tengli	Kim	tInterpreter
$[h, -] \times_v [h, -]$	40%	0%	40%	100%
$[hv, sp](m) \times_v [v, -]$	40%	0%	60%	100%

Table 13. Comparison on Combining Tables whose Primitive Tables are Dissimilar

6. Conclusions

This paper proposes the tInterpreter technique that interprets a table based on its transformation into the corresponding horizontal 1-D tables. Essentially, it utilizes all necessary methods to accomplish the table transformation. These methods are (i) the similarity of two given cells with respect to the data type and the semantic correspondence concerns; (ii) the discovery for the boundary of a primitive table residing in a composite table based on the type similarity; (iii) the identification of the attribute-value relationship and the value association of cells based on their semantic correspondence; and (iv) the integration of two pieces of similar or dissimilar information via the vertical or horizontal manner respectively. The experimental result showed that the overall effectiveness of tInterpreter was higher than Chen, Tengli and Kim as shown in Figure 5. Future work focuses on (i) the interpretation of tables involved with String values; and (ii) the Web query form analysis.

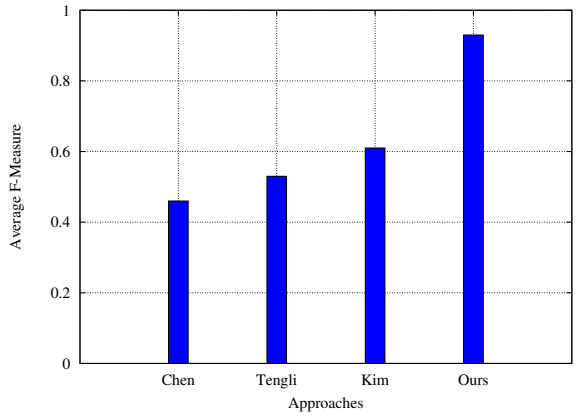

Figure 5. The Overall Comparison on 14 Tables

References

[1] Embley, D.W., Hurst, M., Lopresti, D.P., Nagy, G. (2006). Table-processing paradigms: a research survey. IJDAR, 8 66–86.

[2] Tengli, A.,Yang,Y., Ma,N.L. (2004) Learning table extraction from examples. *In*: COLING '04: Proceedings of the 20th international conference on Computational Linguistics, Morristown, NJ, USA, Association for Computational Linguistics, 987.

[3] Yang, Y., Luk, W.S. (2002). A framework for web table mining. *In:* WIDM '02: Proceedings of the 4th international workshop on Web information and data management, New York, NY, USA, ACM. 36–42.

[4] Zanibbi, R., Blostein, D., Cordy, J. (2003). A survey of table recognition: Models, observations, transformations, and inferences, *International Journal of Document Analysis and Recognition* 7. 1–16.

[5] Chen, H.H., Tsai, S.C., Tsai, J.H. (2000). Mining tables from large scale html texts. *In*: Proceedings in International Conference on Computational Linguistics.

[6] Embley, D.W., Tao, C., Liddle, S.W. (2005). Automating the extraction of data from html tables with unknown structure, *Data Knowl. Eng*. 54 3–28.

[7] Kim, Y.S., Lee, K.H. (2008). Extracting logical structures from html tables. *Comput. Stand. Interfaces,* 30. 296–308.

[8] Wang, Y., Hu, J. (2002). A machine learning based approach for table detection on the web. *In*: WWW '02: Proceedings of the 11th international conference on World Wide Web, New York, NY, USA, ACM 242–250.

[9] Li,S., Liu, M., Peng, Z. (2004). Wrapping html tables in to xml. *In*: WISE. 47–152.

[10] Kim,Y.S., Lee, K.H. (2005). Detecting tables in web documents, *Engineering Applications of Artificial Intelligenc*e 18. 745–757.

[11] Miller, G.A. (1995). Wordnet: A lexical database for English. *Communications of the ACM* 38 39–41.

[12] Tansalarak, N., Claypool, K.T. (2007). Qmatch - using paths to match xml schemas. *Data Knowl. Eng*. 60. 260–282.

[13] Pinto, D., Branstein, M., Coleman, R., Croft, W.B., King, M., Li, W., Wei, X. (2002): Quasm: A system for question answering using semi-structured data. *In*: In Proceedings of the Joint Conference on Digital Libraries (JCDL) 200. 246–55.