An Algorithm for Documenting Relationships in a Set of Reports

Roger L. Goodwin
USDA, National Agricultural Statistics Service, 3251 Old Lee Highway, Room 301, Fairfax, VA 22030
Email: Roger_Goodwin@dc-sug.org

***Abstract*— A typical tabular business report contains a set of cells. The cells may contain raw numeric values, character labels, and formulas. This paper will present a bottom-up algorithm for visually documenting the cells in a set of tabular reports. The algorithm draws a tree-like structure in a view port. The nodes in the tree can represent the either a raw numeric value or a formula. The arcs in the tree show which nodes share common variables in their respective formulas. In a requirements driven environment, this visual documentation allows the programmer and the system analyst to trace the final programmed reports (i.e. the final product) back to the requirements documentation.**

***Index Terms*— Business planning, documentation, visual languages, trees (graphs), hidden lines**

I. Introduction

Statistical agencies conduct follow-up surveys to obtain additional data on survey respondents for a variety of reasons. Some of these reasons might be to clarify a respondent's answers to the original survey, to measure the reasons for non-response, to measure the amount of duplicate data, etc. Consider the problem of tracking the survey's cost and progress. Management must monitor the costs of the survey on a routine basis to ensure that the actual expenditures do not exceed the budgeted dollar amounts. Management must monitor the progress of the survey on a routine basis to ensure that the work will be completed on time.

Reference [3][1] described a complex programming environment in which a set of reports needed to be implemented. The programmer had to implement 10 reports that contained 173 cells where 7 systems were polled weekly for the information. We documented the report layouts and sources of the information for the cells in the reports using the Project Management Body of Knowledge (PMBOK). We had 127 pages of formal documentation and no visual documentation. Section II will give some background information on the reporting environment. Section III will give details on the reports. Reference [3] explained how to construct a tree using the cell references in the tables for additional documentation purposes. However, the paper did not include an algorithm. This paper will begin with three tabular reports, then develop a syntax that is suitable for an algorithm written in Visual Fortran (or any other complier for the matter) to recognize. The algorithm must recognize four situations:

1) A raw value in a report.
2) A label instead of a raw value in a report.
3) A calculation in a report.
4) A relationship between two reports.

In this paper, an algorithm will be outlined. Three modules will be presented using Visual Fortran.

- `Main:` After the set of tables are read into a matrix of records, the cell type gets determined by a user-defined function called `celltype()`. Then, the `Main` module calls another module to draw those cells. See the next item on `Draw4`.
- `Draw4:` This module draws cell relationship 4 defined in Section III-A. Cell relationship 4 is both a calculation and a relationship cell between two reports. The remaining three cell relationships have been omitted, but are much simpler to draw given this one.
- `Hidden:` This module detects when a line has been covered-up (or hidden) behind a node. This is undesirable in the visual program being presented. We could identify the situation by simply changing the line style. Alternatively, we could also perform some simple calculations and move nodes around until the lines become fully visible. This will be discussed in Section VI.

Finally, Section VII gives some run-time analyzes for the above modules. The Appendix at the end of the paper shows how the output should look for three of the reports.

II. Reporting Environment

The Cost and Progress (C&P) system polls other systems for data. The data is summarized and displayed to the survey managers. The system is developed and maintained by Decennial Management Division (DMD) staff. It is used to track the actual survey costs and the progress of the survey. The progress of the survey is typically measured by the number of questionnaires received back. These questionnaires may be completed interviews as well as refusals and non-responders. Figure 1 shows the flow

[1]Portions reprinted, with permission, from R. L. Goodwin, "Conveying Cell Relationships in a Reporting Environment," IEEE Southeastcon Conference Proceedings, Richmond, VA, 2007, pp 87-92, ©2007 IEEE.

© 2008 ACADEMY PUBLISHER

of the forms during the operation. The majority of the payroll costs in this system occur during the interviewing phase. Overall, we are going to track two items.

- Production payroll costs.
- Questionnaires.

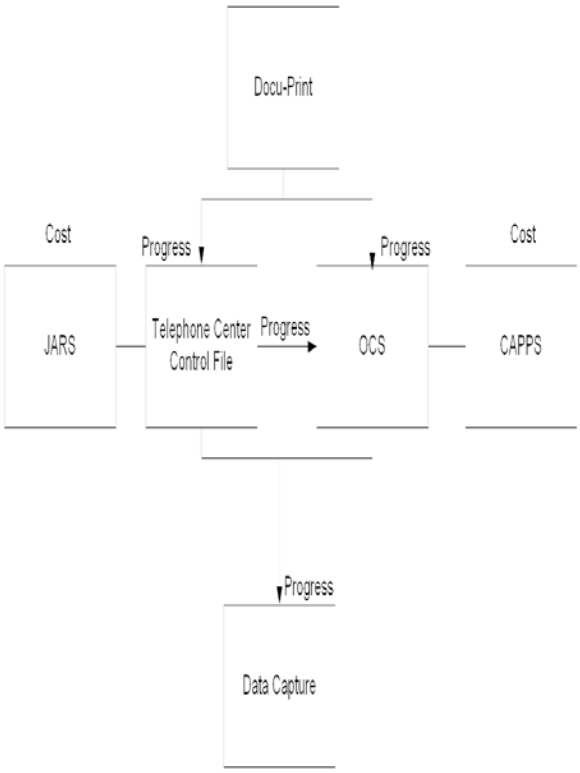

Figure 1. This figure shows how the questionnaires are tracked. Questionnaires can be tracked either within a system or by a control file. Direct payroll costs are incurred during the interview phase at the telephone center and in the field. Payroll costs are directly proportional to the workload (i.e. the number of questionnaires). ©2007 IEEE

The next two sections briefly describe each system. Section II-A describes the systems that track costs. Section II-B describes the systems that track progress. See Figure 1 for an overview. A survey will not even enter the planning stage without a budget (or a future commitment) from the Decennial Budget Branch. Once a survey has been funded and planned, one of the next steps is to print the follow-up questionnaires. These questionnaires will either be sent to the telephone centers or to the field offices, or both (in sequential order). Telephone clerks and enumerators will contact respondents according to the survey protocols. Upon completion, the questionnaires are sent back to the data capture center for check-in and further processing. Once the questionnaires have been received and keyed at the data capture center by the clerks, the C& P System work ends. During the enumeration period, the employee payroll is summarized and posted to the C& P System. Although the names of the clerks and enumerators are of fundamental value to their immediate supervisors, such detailed data are not necessary to track the cost of the survey.

The reports are complex as seven systems are being polled each weekend. Some reports are simple while some are very detailed. The data collected in the C&P System can be used to create more technical information such as calculating the average cost to follow-up on a non-interview case, and the average cost of the interviewed case, the variance of these costs, etc.

A. Tracking Costs

This section lists the systems and/or files used for tracking the direct costs of the survey.

1) DMD Cost Models — The cost models contain *budgeted* dollar amounts; with varying scenarios as to staffing, workloads, etc. Some of the models include overhead costs (such as holiday pay, and other benefits) while some do not. Traditionally, the C&P System uses the direct cost models without the overheads. The term 'allocated' in this specification is used in the general sense. The dollar figure may or may not include unfunded requests, and some funding may or may not appear in the operating plans, depending on where our requests are in the approval process. *The cost models are developed in advance, before the survey begins.*
2) Jefferson Activity Reporting System (JARS) — The JARS system contains actual payroll data for telephone center employees. The C&P System summarizes the JARS data to obtain direct telephone center payroll costs. JARS reports actual payroll hours and direct salaries; no overheads and no benefits.
3) Census Automated Personnel and Payroll System (CAPPS) — The CAPPS system tracks field employee payroll costs incurred during the interview phase of the survey. The CAPPS system contains actual payroll data and reimbursable data for field employees. The C&P system summarizes the CAPPS data to obtain direct field costs.

B. Tracking Progress

This section lists the systems and/or files used for tracking the progress of the survey.

1) The Docu-Print System —- This is where the forms are printed. Obtain counts of the questionnaires printed for the survey here — exclude miss prints, dummy-fill questionnaires, etc.
2) Telephone Center Control File — This is one of many control files. This particular control file contains the IDs of the questionnaires received by the telephone center. The completed questionnaires may be returned to the data capture center or may by recycled for a field follow-up interview (but not vise-versa).
3) Operational Control System (OCS) — This system contains the IDs of the forms that the Field (FLD) offices have received. Among other things, this database contains the outcome codes for each questionnaire sent for personal visit followup. The

© 2008 ACADEMY PUBLISHER

C&P system summarizes the data to obtain counts of active cases, completed cases, and non-interview cases using outcome codes. The C&P System sums the number of active questionnaires and completed questionnaires to obtain the total number of questionnaires.

4) Data Capture — The data capture center is the final destination for the questionnaires in the survey. The check-in file contains all of the questionnaire IDs. The questionnaires are checked-in, imaged, and keyed by clerks.

III. The Reports

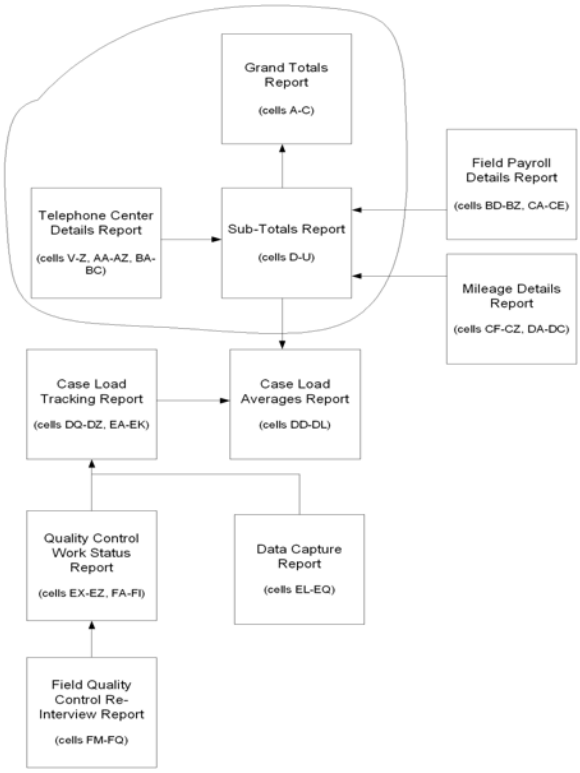

Figure 2. This figure contains ten reports. The *Grand Totals Report* contains high level summary information about the survey. Its four subordinate reports contain more details about the grand totals in regards to the dollars spent. The cell references are denoted by letters in parentheses for each report. The relationships of the cells to each other are not depicted in this figure.©2007 IEEE

Fig. 2 displays ten business reports for tracking the cost and progress of a survey from a management's perspective. Fig. 2 also shows the hierarchy of ten reports. For brevity, this paper will concentrate on the three reports that are circled. The three reports listed below have cell relationships with-in them and between them. The `Grand Totals` report has two subordinate reports. The cell references appear in letters in parentheses. The systems being polled appear outside of the parentheses. Each report has a set of column and row titles. We wish to show the cell relationships and the between-cell relationships graphically.

Grand Totals Report:

Total Dollars Budgeted	Total Direct Costs	Percent Direct Costs
Bud (A)	CAPPS/JARS (B)	Derived (C)

Sub-Totals Report:

	Dollars Budgeted	Direct Costs	Percent Direct Costs	Hours Budgeted	Hours Used	Percent Hours Used
Telephone Centers:	Bud (D)	JARS(E)	Derived(F)	Bud (G)	JARS(H)	Derived(I)
Field:	Bud (J)	CAPPS(K)	Derived(L)	Bud (M)	CAPPS(N)	Derived (O)
Totals:	Bud (P)	Derived(Q)	Derived(R)	Bud (S)	Derived(T)	Derived (U)

Telephone Center Details Report:

	Dollars Budgeted	Direct Costs	Percent Direct Costs	Hours Budgeted	Hours Used	Percent Hours Used
Phone Intvrs	Bud (V)	JARS (W)	Derived (X)	Bud (Y)	JARS (Z)	Derived (AA)
Coaches/Montrs	Bud (AC)	JARS (AD)	Derived (AE)	Bud (AF)	JARS (AG)	Derived (AH)
Supervisors	Bud (AJ)	JARS (AK)	Derived (AL)	Bud (AM)	JARS (AN)	Derived (AO)
Othr Employees	Bud (AQ)	JARS (AR)	Derived (AS)	Bud (AT)	JARS (AV)	Derived (AU)
Totals:	xxx (AW)	xxx (AX)	Blank (AY)	xxx (AZ)	xxx (BA)	Blank (BB)

As input to the algorithm, the column and row titles will be removed as well as the systems (for now). That just leaves us with the following three shells. However, for the program to function properly, the derived columns must be revised to reflect those cells that they are calculated from. So, we have the following table shells as the input file.

A	B	C=B/A*100					(ITERATION 9)
D	E	F=E/D*100	G	H	I=H/G*100		(ITERATION 8)
J	K	L=K/J*100	M	N	O=N/M*100		(ITERATION 7)
P=D+J=A		Q=E+K=B	S=G+M		T=H+N		(ITERATION 6)
V	W	X=W/V*100	Y	Z		AA=Z/Y*100	(ITERATION 5)
AC	AD	AE=AD/AC*100	AF	AG		AH=AG/AF*100	(ITERATION 4)
AJ	AK	AL=AK/AJ*100	AM	AN		AO=AN/AM*100	(ITERATION 3)
AQ	AR	AS=AR/AQ*100	AT	AV		AU=AV/AT*100	(ITERATION 2)
AW=V+AC+AJ+AQ=D		AX=W+AD+AK+AR=E	AZ=Y+AF+AM+AT=G			BA=Z+AG+AN+AV=H	(ITERATION 1)

On the bottom line of the input file, the intention of using `AW=V+AC+AJ+AQ` is to reduce the space needed to draw the nodes on the graph. So, we will use `AW` as a label as opposed to the actual calculation. I know that my total dollars budgeted for the Telephone Center on the Details Report (cell `AW`) must equal to cell `D` on the Sub-Totals Report. However, this relationship does not exist between the reports. How do I include it in the input file? By creating the syntax, where if, there are two equal signs such that `<label>=<calculation>=<between report cell>`, then `<between report cell>` reference connects two tables.

The "ITERATION" numbers listed to the far right are reference numbers that will be explained later in the paper. The use of the underscore "_" is arbitrary to delineate the three reports. Once the algorithm reads two or more delimiters together, then it knows that it is at the bottom of a new report (reading from the bottom up). The reader could easily use some other character. Knowledge of prefix, postfix, or infix evaluation algorithms is not needed in this paper. Recall that these are simple business reports with simple totals and percentages. We are creating visual documentation. Knowledge of stacks and arrays at an undergraduate level is all that is required.

© 2008 ACADEMY PUBLISHER

A. Cell Syntax

The cell content in a report can vary depending on if it is a direct cost, a direct amount of hours used, a total, a percentage, etc. Our cell syntax can have one of the following forms.

1) <cell> which represents a direct cost or a direct amount of hours used in the report. Use the cell reference in the graphical representation.
2) <label>=<cell> which represents a direct cost or a direct amount of hours used in the report. Use the label in the graphical representation.
3) <label>=<calculation> which represents a calculation in the report, usually a percentage or a sub-total. Use the label in the graphical representation.
4) <label>=<calculation>=<between report cell> which represents a calculation in a report *and* a relationship between two reports. Use the label in the graphical representation.

IV. Implementation Details

The algorithm for drawing the visual documentation of the three reports in Section III is fairly straight forward. The business reports given in ref [3] have the typical columnar form, with sub-totals at the bottom and percentages to the right of a set of columns. Using a *bottom-up programming technique*, we begin with the last report on the input file.

1) Select last report in the list.
2) Begin with the bottom line. Place the calculations on the graph. If labels are specified, place the labels in replace of the calculations.
3) At the end of the line, place the operands beneath the labels (or calculations) as close as possible to its label (or calculation) on the graph. Draw an arc from each operand to the label (or calculation).
4) Store any between report cell relationships (pairs) encountered in an array. Hold them until the entire set of reports have been placed on the graph.
5) Delete line.
6) Goto Step (1) until no more lines in the report.
7) Draw arcs for relationships between reports.
8) Verify operands are in the table. If missing, output error message.

Ideally, we place our nodes in steps (2) and (3) approximately close to each other so that when the arrows have to be drawn, we do not cross arcs. This may not always be possible. It should be minimized to avoid a cluttered tree. Placement of the nodes is rather arbitrary — particularly at the onset of the algorithm. However, since this is visual documentation, centering statements can be easily worked into the algorithm. The reader may have noticed that some of the information in the input tables is redundant. For example, with the label AW appearing in the tree, we are merely verifying that the nodes V, AC, AJ, and AQ appear in the table somewhere. When they are read, they can be discarded. A similar situation occurs with the labels AX, AZ, and BA and their nodes.

The programmer does not have to add scroll bars to the visible window — at least in Visual Fortran. The viewport has the same dimensions as the window. It is probably best not to make the viewport smaller than the window. Otherwise, clipping may occur. We do not want hidden lines behind the nodes to occur. So, some minimal calculations must take place. Detecting hidden lines will be discussed in Section VI. This entails saving the coordinates of the nodes in the window. To implement the visual documentation for the tables using Visual Fortran, we need the following built-in procedure and function calls.

- SETTEXTPOSITION(x, y, t) — Moves to the (x, y) coordinate on the computer screen. The record t contains the coordinate position of the text previously displayed on the screen. This procedure only affects the OUTTEXT and ELLIPSE procedures.
- OUTTEXT("message") — Writes the cell reference letter(s).
- ELLIPSE($GBORDER, x1, y1, x2, y2) — Draws the circle where (x_1, y_1) are coordinates of the upper left-hand corner of the circle and (x_2, y_2) are the coordinates of the lower right-hand corner of the circle.
- MOVETO(x,y, xyt) — Similar to the SETTEXTPOSITION procedure, it moves to the (x, y) coordinate on the computer screen. This procedure only affects the LINETO function.
- LINETO(x,y) —- Draws a directed line (an arrow) from the current position to the (x, y) coordinates.

With the above Visual Fortran procedure and function calls, and our usual Fortran 90 loop and data structure constructs, we can proceed to the pseudo code in Section V.

V. Pseudo Code

This section outlines some of the pseudo Visual Fortran code for implementing the automated documentation for tables. The record table.lines() contains the three reports. In the procedure main loop, we loop through each report line, classifying each cell by its type. Based on the cell type, a procedure is called to draw it.

```
main
    ! loop through the report lines
    do i ← max_lines to 1 by −1
      ! loop through the cells
      do j ← 1 to max_cells
        cell ← table.lines(i, j)
        ! is cell a table delimeter?
        if cell_type(cell) ≠ 0 then do
      - if celltype(cell)= 4 then call Draw4
      - if celltype(cell)= 3 then call Draw3
      - if celltype(cell)= 2 then call Draw2
```

© 2008 ACADEMY PUBLISHER

- if celltype(cell)= 1 then call Draw1

end main

Two lists are maintained. One list keeps track of the nodes that have already been drawn on the screen and it is called `s_list`. It contains the node labels along with their (x, y) coordinates. This is useful for identifying the following situations.

- Neither node appears on the screen. Draw two nodes and one arrow.
- One node already appears on the screen. Draw one node and one arrow.
- Both nodes already appear on the screen. Draw one arrow.

In addition, the list `s_list` is useful for finding hidden lines. Hidden lines will be discussed in Section VI.

The other list is called `b_list` and it contains those nodes that represent between-report relationships outlined in the syntax in Section III-A. The following pseudo code is used for the `Draw4` routine. The node on the far right in the cell is the between-report relationship cell and will be identified first, and added to the `b_list`. Since the general structure of the cell is known, we can proceed from left to right afterwards.

draw4
- $k \leftarrow max$!max cell size of array
- do while cell(k) $\neq$ blank $k \leftarrow k - 1$
- nmax $\leftarrow k$
- add cell(k) to b_list !first "between" report pairs
- $k \leftarrow 1$

  if cell(k) not in s_list then !first cell to the left
  - call SETTEXTPOSITION
  - call OUTTEXT
  - call ELLIPSE
  - add cell(k) to s_list
  - add cell(k) to b_list !second "between" report pairs

  else call SETTEXTPOSITION
- $k \leftarrow 2$

  do while $k \leq nmax$
  - if not operator(cell(k)) then

    if cell(k) not in s_list then
    - call SETEXTPOSITION !below
    - call OUTTEXT
    - call ELLIPSE
    - call MOVETO
    - junk = LINETO
    - add cell(k) to s_list

    else
    - call MOVETO
    - junk = LINETO

  end do

end draw4

As one can imagine, most of the work from these drawing subroutines will be moving around on the screen and either writing text, drawing a circle, or drawing an arrow. Much overhead in terms of maintaining x and y coordinates is required. The output from the methods outlined in this paper appears in the Appendix on page 8. The iterations listed on the right-hand side are from the outer loop in the `main` procedure. To obtain this output, several decisions were made to keep the algorithms easy to program. The optimal placement of nodes has not been calculated. The optimal placement of nodes involves minimizing the amount of hidden lines. Adding a constant (or subtracting depending on if it is x or y) to a coordinate is much simpler to program. The reader can still merely detect a hidden line, and change the line style without trying to reposition any nodes. Reference [5, p. 89] gives the following masks for line styles.

Style	Mask	Internal Windows
Solid	#FFFF	PS_SOLID
Dash	#EEEE	PS_DASH
Dot Dot Dot	#AAAA	PS_DOT

VI. Hidden Lines

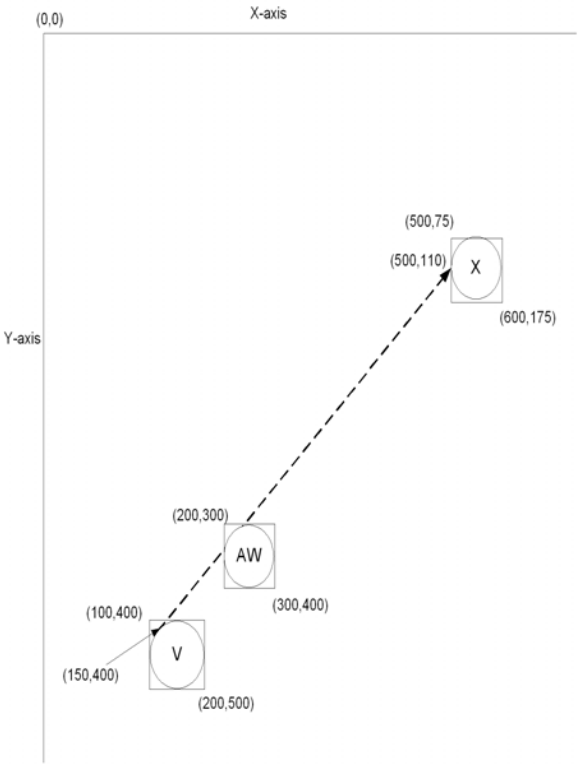

Figure 3. This figure shows a numerical example of a hidden line. The line passes through node `AW` and cannot be seen. To avoid this, node `X` should be moved to the left until the line is out of the invisible plane of node `AW`.

Hidden lines can be deduced. Take the simple example in Fig. 3. Given two points, we can use analytic geometry find the equation of a line using the *point-slope formula* in Equation 1.

$$y - y_1 = m(x - x_1) \tag{1}$$

where the slope $m = \frac{y_2 - y_1}{x_2 - x_1}$. The end points of our line are $(150, 400)$ and $(500, 110)$. The slope m is

© 2008 ACADEMY PUBLISHER

$m = \frac{110-400}{500-150} = -\frac{290}{350}$. The slope is negative even though the line appears to have a positive slope because the upper-left corner of the window is the origin $(0, 0)$. So, the equation of the line is $y = -\frac{29}{35}x + \frac{3670}{7}$ subject to $150 \leq x \leq 500$, and $75 \leq y \leq 400$. The plane containing the node `AW` is bounded by four lines — each of which has a linear equation. If any of these four lines intersect with our line, then our line is hidden by the plane `AW`. The upper right corner coordinate of the plane `AW` is $(300, 300)$. So, using Equation 1, we obtain $y = 300$ subject to $200 \leq x \leq 300$. It's easily verified that the two lines intersect at $(201.3, 300)$. So, the node `X` needs to be moved to the left until the system of equations is not satisfied. Finding the point of intersection of two lines is hard work and not necessary. This will be discussed in the next paragraph.

Reference [2] gives a discussion of topic of hidden lines and the mathematics for removing them in 3D. Our problem is much simpler. We can formally state this problem and it's solution using Linear Algebra. Then, write a short Visual Fortran function for determining when two truncated linear equations intersect. The *determinant* of two linear equations will let us know if a unique solution exists. Reference [4] gives the determinant as $a_{11}a_{22} - a_{12}a_{21}$ for the system of equations:

$$a_{11}x_1 + a_{12}x_2 = b_1$$
$$a_{21}x_1 + a_{22}x_2 = b_2$$

So, we re-write our two point-slope equations to get

$$35x_1 + 29x_2 = 18350$$
$$1x_1 + 0x_2 = 300$$

The bounds to the line and the top line on the `AW` node are given next.

$$\text{Line\_}x_1 = 150 \leq x \leq 500 = \text{Line\_}x_2$$
$$\text{Line\_}y_1 = 110 \leq y \leq 400 = \text{Line\_}y_2$$
$$\text{Node\_}x_1 = 200 \leq x \leq 300 = \text{Node\_}x_2$$
$$\text{Node\_}y_1 = 300 \leq y \leq 300 = \text{Node\_}y_2$$

In our example, the determinant is $35 \times 0 - 29 \times 1 = -29 \neq 0$. Therefore, the solution is unique (in the case of infinite lines). Next, we must check the bounds. Does the line lie in the bounds of the line of the `AW` node? Yes. So, we again conclude that the line from node `V` to node `X` is hidden by node `AW`. But the reasoning as to why is different than in the preceding paragraph. We did not need to find the point of intersection this time.

The following Visual Fortran function can be used to determine if a line is hidden by a given node.

```
function hidden(Line_x1, Line_y1, Line_x2, Line_y2,
Node_x1, Node_y1, Node_x2, Node_y2)
  - hidden = .false.
  - ! do the given coordinates make sense?
  - if Line_x1 > Line_x2 then swap the two
  - if Line_y1 > Line_y2 then swap the two
  - if Node_x1 > Node_x2 then swap the two
  - if Node_y1 > Node_y2 then swap the two
  - Calculate slope
  - Calculate a11, a12, a21, a22
  - Calculate the determinant
    if (determinant > 0) then
    if (Node_x1 > Line_x1 and Node_x2 < Line_x2)
    and
    ( Node_y1 > Line_y1 and Node_y2 < Line_y2)
    then
    - hidden = .true.
return hidden
⋮
answer = hidden(150, 400, 500, 110, 200, 300, 300,
300)
```

If this function returns a value of TRUE for any one of the four lines associated with the square box around the node, move the node that the line is being drawn to. In our example, we would move the node `X` to the left. The function should be modified by the reader where the user only inputs the upper left-hand corner and lower right-hand corner of the box to the node and calculates the four lines associated with the node in question. This would take some extra programming effort. Also, for practical purposes, this function should be executed *before* a line is drawn. Otherwise, the line would have to be erased by re-drawing it with the background color of the computer screen.

VII. Some Simple Algorithm Analyzes

The problem presented in this paper is to draw the relationships amongst the cells in a set of reports. This is done by drawing a tree-like structure. How much work is performed by the procedures presented? The following notation will be used in this section. Let l be the total number of lines for all of the reports. Let c be the total number of cells for all of the reports. Let n be the total number of nodes. Let o be the total number of operators. Since we have four different cell types, some kind of distribution will have to be assumed to be able to determine how many times each of the drawing procedures will be executed. With some insight from the pseudo code, the following distribution will be assumed.

$$P(\text{draw cell type } i) = \begin{cases} \frac{1}{10}, & \text{if cell type } i = 1. \\ \frac{1}{10}, & \text{if cell type } i = 2. \\ \frac{6}{10}, & \text{if cell type } i = 3. \\ \frac{2}{10}, & \text{if cell type } i = 4. \end{cases}$$

The distribution adds up to 100%. Now for the explanation why this distribution is skewed, and why most of the probability is at cell type 3. Cell types 3 and 4 contain the calculations. Thus, once those cells have been parsed and placed onto the window, then, most likely cell types 1 and 2 do not even need to be drawn. They are retained for error checking. Cell type 3 has a much larger probability than cell type 4 because it is not anticipated that a large amount of the calculations will be directly related to other report totals. Just as with the within-report relationships,

© 2008 ACADEMY PUBLISHER

we kept the number of lines to a minimum; we also wish to keep the number of between-report relationship lines to a minimum.

A. `Main`

The `Main` procedure loops through the report lines and each cell on the lines. We expect to do at most $l \times c \times (n + o)$ iterations. Assuming two directed lines per node, $2n$ lines need to be drawn. How much movement on the screen takes place? Placing the nodes does not require moving to any particular position. But, placing text in the viewport and a line in the viewport does. So, the movement on the screen will be directly proportional to the number of lines that need to be drawn and the number of nodes (labels alway accompany nodes).

B. `Draw4`

The `Draw4` procedure has a loop construct in it. It draws those cells that have cell relationships between reports. In addition, the procedure must be executed a number of times by the `Main` procedure. So, first let's look at the loop construct. Consider that a cell can have a maximum of $o + n$ characters. This includes the o operators and n operands. The loop will do at most $(o + n - 4)$ iterations between the two equal signs for a cell of type 4. Then, in the `main` procedure, the `draw4` procedure will be executed $l \times \frac{2c}{10}$.

C. `Hidden`

The `Hidden` function has no looping construct in it — just a group of statements executed in sequence. It identifies whether or not a line will be hidden behind a cell or not. To avoid hidden lines in the tree, this function must be executed before each node is placed in the viewport (window). The function executes, at a minimum, when two nodes are in the window and more nodes need to be drawn. Each node in the list `s_list` must be examined for a potential hidden line problem to take appropriate action. So, this procedure gets executed $\frac{n(n+1)}{2} - 2$ times.

VIII. Concluding Remarks

This paper outlined a simple procedure for creating visual documentation for tables. Most of the work performed by the modules involves moving around in the viewport (window), and drawing lines, circles and writing labels. When the reports become complex, and the nodes have been placed in the window, then the issue of hidden lines occur. This can be an easy situation to program around. Simply detect the hidden lines and change their style. An alternative and more complex program would detect the hidden lines and move nodes until the hidden lines are fully visible. The latter solution requires an extra programming effort.

References

[1] M. Etzel and K. Dickinson, *Digital Visual Programmer's Guide*, Digital Press, Woburn, MA, 1999.

[2] J. D. Foley and A. Van Dam, *Fundamentals of Interactive Computer Graphics*, Addison-Wesley Publishing Company, Reading, Massachusetts, 1982.

[3] R. L. Goodwin, "Conveying Cell Relationships in a Reporting Environment," IEEE Southeastcon Conference Proceedings, Richmond, VA, 2007, pp 87-92.

[4] S. I. Grossman, *Elementary Linear Algebra (Second Edition)*, Wadsworth Publishing Company, Belmonth, California, 1984.

[5] N. Lawrence, *Compaq Visual Fortran: A Guide to Creating Windows Applications*,Digital Press, Burlington, MA, 2002.

Biography

Roger Goodwin started his civil service career at the US Census Bureau in 1999. At the Census Bureau, Mr. Goodwin worked in the Decennial Directorate and the Economic Directorate as a Mathematical Statistician. Currently, Mr. Goodwin works for the National Agricultural Statistics Service at the US Department of Agriculture. He received his BS in Computer Science in 1988 from Old Dominion University, his MS in Applied Statistics in 1998 from Old Dominion University, and his certification in Software Engineering Processes in 2005 from Learning Tree. He has been the SAS Liaison for the DC SAS User's Group (DCSUG) since 2003. Mr. Goodwin is a member of the Society for Industrial and Applied Mathematics (SIAM), the Institute of Electrical and Electronics Engineers (IEEE), and the Mathematical Association of America (MAA). He has presented papers within the agencies that he has worked for, at the Northeast SAS User's Group conferences, at SIAM conferences, and at IEEE.

© 2008 ACADEMY PUBLISHER

APPENDIX

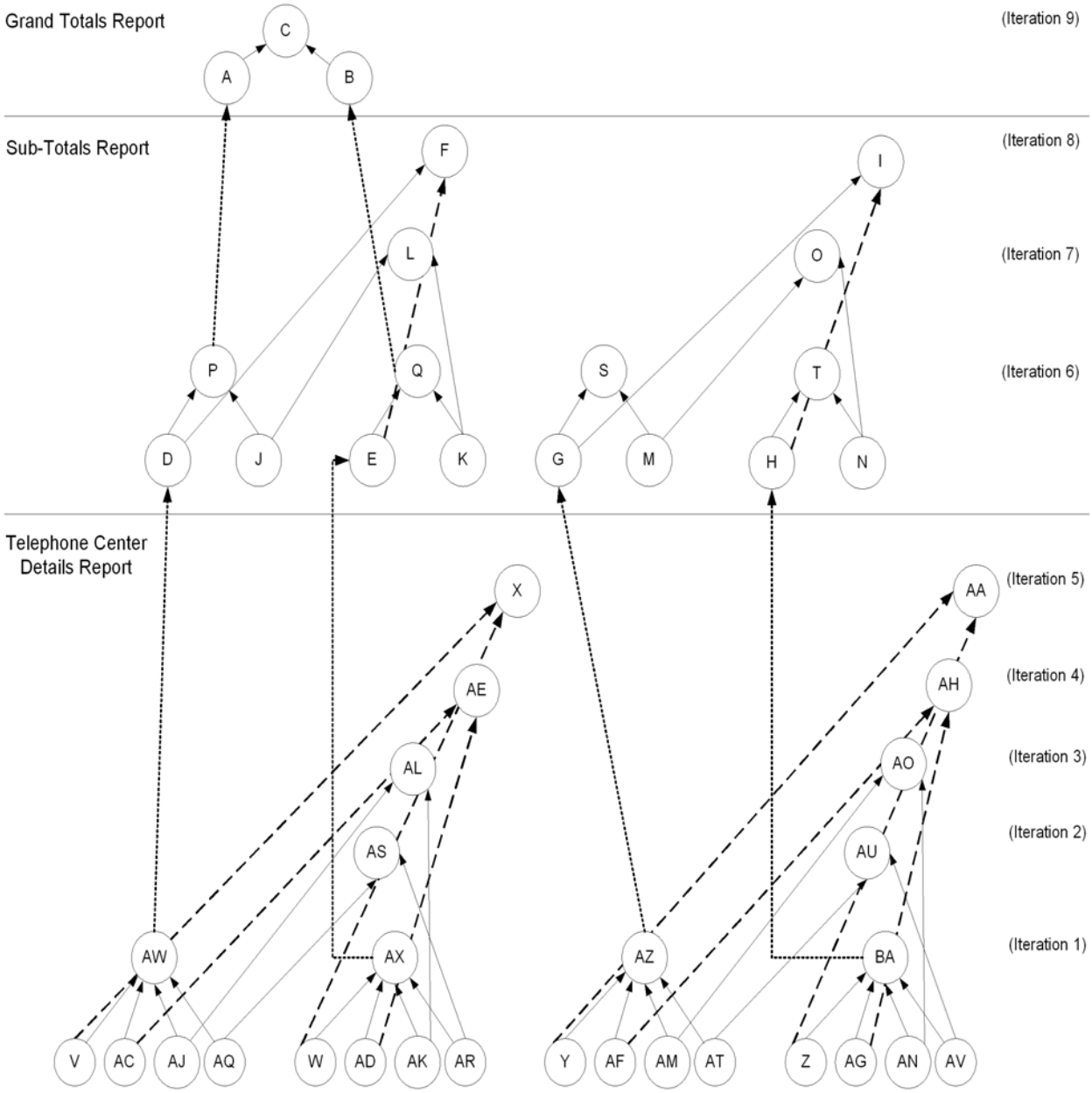

Figure 4. This figure shows the output from the pseudo code. The arrows pointing to the nodes show that those nodes are referenced by that cell. The heavy dotted lines denote the relationships between the reports. The heavy dashed lines denote when hidden lines occurred. The hierarchy nature of the cells in the reports has been preserved. The arrows tend to go in a single direction and have been minimized.

© 2008 ACADEMY PUBLISHER