

# **AC 2007-1807: SPATIALLY RECURSIVE SPREADSHEET COMPUTATIONS: TEACHING THE CRITICAL PATH METHOD OF SCHEDULING USING TWO-DIMENSIONAL FUNCTION RANGES VERSUS TRADITIONAL ONE-DIMENSIONAL OBJECT-ORIENTED PROGRAMMING**

## **Gunnar Lucko, Catholic University of America**

Gunnar Lucko, Ph.D. is an assistant professor and director of the Construction Engineering and Management program in the Department of Civil Engineering at The Catholic University of America. His research interests include network scheduling, construction operations simulation and optimization, equipment economics, constructability analysis, and innovative teaching methods. He has studied statistical equipment valuation models and has participated in research for the Construction Industry Institute and the National Collegiate Inventors and Innovators Alliance. His e-mail address is <lucko@cua.edu> and his web address is <<http://engineering.cua.edu/faculty/profiles/lucko.cfm>>.

## **Michael Madden, United Space Alliance**

Michael G. Madden, M.S.T.M. has over 24 years of experience at United Space Alliance, the space shuttle operations contractor for the National Aeronautics and Space Administration. He has planned and scheduled the ground processing operations for both OV-105 Endeavour and OV-104 Atlantis as Senior Vehicle Engineer. More recently, as Project Leader III he is team leader for developing simulation models and decision support systems for the orbiter processing facility. As Senior Engineering Liaison he is providing processing, data mining, and specifications expertise for the Lockheed Martin crew exploration vehicle proposal on the next generation human spaceflight program. He is a doctoral student in Industrial Engineering at the University of Miami.

## **Justin Molineaux, Catholic University of America**

Justin P. Molineaux is an undergraduate student in the Department of Electrical Engineering and Computer Science at The Catholic University of America.

# **Spatially Recursive Spreadsheet Computations: Teaching the Critical Path Method of Scheduling using Two- Dimensional Function Ranges versus Traditional One- Dimensional Object-Oriented Programming**

## **Abstract**

Project management is the art and science of planning and controlling projects in their various aspects of time, cost, and scope. Scheduling focuses on the time aspect while considering the various needed resources. The critical path method (CPM) is the most common scheduling technique, whereby the project is broken down into activities with specific durations and relationships among each other. Calculation occurs in two major steps. In the forward pass each activity is scheduled to occur as early as possible while obeying its dependency conditions. The backward pass examines the inverse case of delaying all activities as late as possible without impacting the project end. The flexibility of each activity, its float, is assessed by comparing these extreme cases. Activities with zero float are time critical, as postponing any of them would impact the project end.

Any traditional computer program for the two-step CPM algorithm consists of defining variables for the time and dependency information of each activity from the schedule input, sorting them, making case distinctions whenever the dependency structure splits or merges between predecessors and successors, and saving the maximum early dates and minimum late dates to the output. An object-oriented programming (OOP) approach would use the appealing existing division into objects, the activities, which are related in a clearly defined sequence. It would follow a one-dimensional flow of individual commands including various loop statements to accommodate the case distinctions at forks in the dependency structure. Numerous standard textbooks on project management that have been reviewed fail to consider these case distinctions in their presentation of the CPM algorithm, which in the experience of the authors often leads to students initially having difficulties in how to apply the parallel evaluation of numerous activities under CPM to solve complex schedules.

The authors developed an educational unit for teaching CPM to undergraduate civil engineering students concentrating in construction engineering and management. Students learn using spreadsheet functions and diagrams before CPM is introduced in this course on computer use in construction. In teamwork under the guidance of the instructor, they then develop the CPM algorithm through manual scheduling exercises in conjunction with computer spreadsheet modules for each part of the complete CPM analysis. This “learning by doing” build a deeper understanding of the mechanics of CPM. Finally, commercial scheduling software is introduced.

The modular spreadsheet presents an innovative non-OOP approach to solving CPM schedules of arbitrary complexity through beneficial use of its two-dimensional spatial format. Recursive function ranges in that two-dimensional matrix format allow solving the schedule directly. They are fully scaleable up to the available number of rows and columns in the spreadsheet. Further research will add graphical capabilities that can be controlled by the students with traditional macro programming.

## **Introduction**

Among the most fundamental tasks of construction project managers is planning and controlling a projects with respect to the diverse and complex interplay of various dimensions. These dimensions are typically considered to be time, cost, and scope of the project. Scheduling, the art and science of managing the time aspect of projects, is an active discipline for which graduate-level courses are offered in construction engineering and management programs at universities, which are taught with a plethora of different textbooks (1), the leading Journal of Construction Engineering and Management dedicates a Cost and Schedule specialty area to it to publish current research papers, the Project Management Institute, an professional organization of approximately 180,000 members in 2002 created a special interest group called College of Scheduling, and the available commercial scheduling software over the past two decades has made a strong contribution to the popularity of a particular scheduling technique, the critical path method (CPM) (2). Researchers have regularly examined the use of CPM (3, 4, 5, 6, 7), and have found it to be the dominant scheduling technique used in the construction industry practice.

## **Critical Path Method and its Algorithm**

As reported by Weaver (2, p3-4), the roots of CPM go back to “mid 1956. E.I. du Pont de Nemours (Du Pont) was looking for useful things to do with its ‘UNIVAC1’ computer. (...) The Du Pont team was lead [sic] by Morgan R. Walker, key players from [Remington Rand] Univac were James E. Kelley and John Mauchly. Kelley was the mathematician and computer expert.”

Fundamentally, CPM requires that a project is broken down into activities using the Work Breakdown Structure (WBS) approach. Each activity is assigned a unique duration based on resource productivity and is given relationships with preceding and succeeding activities to form a dependency structure. This structure is commonly shown in a network diagram; initially as activity-on-the-arrow (AOA) notation as used by Kelley et al., where activities are represented as arrow across time and their connections are shown as nodes where they start or finish. The graphical representation of schedules in modern use of CPM prefers the precedence diagramming method (PDM), also known more descriptively as the activity-on-the-node (AON) notation, where activities are boxes and logic links are symbolized by arrows.

The algorithm for the critical path method itself is carried out in two steps. Under the forward pass the activity durations are added in the positive direction along the time axis beginning at the single overall start activity. The assumption is made that all activities occur as early as possible under the given dependency structure. In general, the finish time of a predecessor becomes the start times of all direct successors, whose finish time in turn is determined by adding the duration of the respective successor. At any merging of logic links in the positive direction (i.e., an activity has multiple predecessors), the maximum earliest finish is used as the new earliest start time. Under the backward pass the activity durations are subtracted in the negative direction along the time axis beginning at the single overall finish activity. The assumption is made that all activities occur as late as possible. The start time of a successor becomes the finish time of all direct predecessors, which start time is determined by subtracting the duration of the respective predecessor. At any merging of logic links in the negative direction (i.e., an activity has multiple

successors), the minimum latest start is used as the new latest finish time. The basic equations for the CPM algorithm are given by Equations 1 through 4.

Forward Pass:

$$ES_i + DUR_i = EF_i$$

**Equation 1**

$$ES_j = \text{maximum (all } i) \{EF_i\}$$

**Equation 2**

Backward Pass:

$$LF_i = \text{minimum (all } i) \{LS_j\}$$

**Equation 3**

$$LF_i - DUR_i = LS_i$$

**Equation 4**

where  $ES$  is the earliest start date,  $EF$  is the earliest finish date,  $LS$  is the latest start date,  $LF$  is the latest finish date,  $DUR$  is the duration, activity  $i$  is a direct predecessor to activity  $j$ . By comparing the earliest and the latest dates, the flexibility or “float” of each activity can be derived using Equations 5 and 6.

Critical Path and Float

$$TF_i = LS_i - ES_i = LF_i - EF_i$$

**Equation 5**

$$FF_i = [\text{minimum (all } j) \{ES_j\}] - EF_i$$

**Equation 6**

where  $TF$  is the total float, also called path float, and  $FF$  is the free float, also called activity float, and an activity is considered as being critical if the total float is zero. Critical activities form a critical path through the schedule network, which may branch and change if activity durations change, that gave the method its name. Critical activities together yield the overall project duration, as they form a direct chain from overall start to overall finish. Any delay in them would immediately result in a delay to the overall finish, since by definition the critical path does not have any flexibility of non-work time built into it that could absorb delays.

### **Traditional Teaching of Critical Path Method**

In the experience of the authors, giving students a deep conceptual understanding of CPM scheduling is essential for preparing them as future managers of construction projects. Standard textbooks that are used in teaching scheduling to students in construction engineering and management (8, 9, 10) typically open with a discussion of characteristics of the construction industry, followed by introducing simple bar charts as a method to show graphically the time range across which activities extend. Afterwards, network scheduling is introduced via the AON notation and the CPM algorithm is explained using a small sample schedule of approximately a dozen activities. Other textbooks that focus on computerized scheduling (11, 12, 13) present CPM in a modular form, where the theory about the algorithm is intermingled with descriptions on how to enter the schedule information into a particular commercial software package.

While the educational approach used by all of these authors clearly emphasizes the use of examples that the student should work through, the authors of this paper have observed that the gaining an understanding of the complexity of schedules and how to handle it in CPM is underestimated. Students often find the recursive process of evaluating merging logic links in both the forward pass and backward pass unclear, as it requires considering several possible

values and selecting the correct one to continue the calculations. The case distinctions at each branching within the dependency structure, together with the necessity of evaluating several activities in parallel are the central challenge in learning CPM. Neither the textbooks, whose text remains static, nor the existing commercial software present this actual dynamic nature of the manual computations properly, at a possible detriment to the students' learning experience.

### **Spreadsheet Calculator for Critical Path Method**

The augment the traditional educational approach for CPM, the authors therefore created a new and unconventional tool for teaching CPM that is particularly geared toward undergraduate students, who might often only take one course in construction engineering and management before graduating from their four-year civil engineering curriculum. The educational unit is part of a computer-assisted introductory course at the sophomore level that covers basic project management principles. The part on CPM scheduling begins with introducing the spreadsheet functions and its diagramming capabilities with in-class guided programming exercises. The CPM algorithm is introduced with paper-and-pencil examples of different complexity and is broken down into its major parts. Under the guidance of the instructor, students then develop the algorithm in the spreadsheet software as tabular modules. The modularization of the algorithm modules is also ideally suited to being solved in teamwork by the class as individual homework assignments. Together, the modules constitute the full CPM analysis with all the mathematical capabilities that commercial software offers as well. This learning by doing concept builds insights into the entire mechanics of CPM reminiscent of the mechanistic view for it that had been advocated for by Lucko (14). Only after building this solid foundation in scheduling is the commercial software introduced.

The spreadsheet called *CPM Calculator* consists of modules that each are arranged in tabular format. Note that more modules are necessary than the commonly discussed parts of the algorithm, the forward pass, the backward pass, and the float calculation, as the schedule logic needs to be extracted from the user-supplied activity table. This conversion of the activity table into a successor matrix, or an equivalent predecessor matrix, is commonly neglected if not omitted entirely from scheduling textbooks, whereas it is a fundamental part of understanding the dependency structure of the schedule at the detail level. The spreadsheet approach of the *CPM Calculator* is innovative in its programming because it lets the students make beneficial use of the two-dimensional spatial format of the spreadsheet. Each module is a two-dimensional matrix that calculates all possible solutions for its part of the algorithm from which the correct solution is selected and displayed before being used further in the next step of the calculation, i.e., the following module. While computationally not optimally efficient, the spreadsheet thus shows openly the flow of how all input values are combined with each other and how new values are created from them. Table 1 shows the fully commented spreadsheet formulas for CPM as successfully implemented and tested by the authors. Note that the cell references depend on the actual location of each matrix table in the spreadsheet and might differ if recreated.

For comparison with the normal computational approach to CPM, the algorithm has also been implemented in object-oriented programming (OOP) code as described further below, under the assumption that the basic element of a project schedule, the activity, lends itself particularly well to OOP. Such computer program consists of creating the activities as objects with a set of

variable attributes for the time and dependency information, sorting them, updating start and finish values until the last entry of an attribute value is retained and not overwritten anymore, thereby automatically accounting for all case distinctions wherever the dependency structure branches, and saving the numerical results to the output. However, while computationally more efficient, the CPM calculation is very difficult to follow in OOP, since programming code in an OOP language updates the numerical values attributed to any object continuously without revealing the inside workings of the code once it has been implemented for execution.

**Table 1: Spreadsheet Formulas for Critical Path Method**

Module	Function	Commands and Explanation
Input	Checks if activity has successors	=IF(OR(D6="",D6="N/A"),"no","yes") Gives "yes" if successors cell contains any value.
Successor Matrix	Extracts activity logic into matrix	=IF(ISERROR(SEARCH(A\$6, \$D6,1)),0,1) Gives "1" if activity name is contained in successor cell.
Predecessor Matrix	Is transposed of successor matrix	=INDEX(Successors,A\$24,\$Q26) Swaps row and column and gives value from successor matrix.
Column Allocation	Sorts activity into column by sequence	=MAX(A\$48:A\$62)+1 =IF(A7=1,HLOOKUP(\$A6,Col_Allocation,3,FALSE),0) If successor matrix contains "1", looks up the activity number and its current column values, selects the maximum value from table and adds one. Function is recursive.
Forward Pass	Propagates maximum early finish for each activity	=IF(A7=1,\$T70,"") =MAX(A70:A84) =HLOOKUP(A6,Forward_Pass,2,FALSE) If the successor matrix contains "1", looks up early start values for the activity and selects the maximum value from table. Function is recursive.
Backward Pass	Propagates minimum late start for each activity	=IF(P7=0,MAX(\$T\$70:\$T\$84),MIN(A89:O89)) =IF(A7=1,VLOOKUP(A\$67,Late_Start,6,FALSE),"") If successor matrix contains "1", looks up late starts and selects minimum value from table. If no successor, selects maximum value of late finishes from table. Function is recursive.
Total Float	Difference of early and late dates	=V89-S70 Deducts the early date from the late date, either for start dates or for finish dates.
Free Float	Difference of minimum early start of successors and early finish	=IF(OR(Results!D6=0, D6="",D6="N/A",D6="-"),0,MIN(A109:O109)-T70) =IF(A7=1,VLOOKUP(A\$67,Early_Start,3,FALSE),"") Looks up early starts and selects minimum value from table. If successor matrix contains "1", looks up activity number and selects its early finish.

## ***Named Ranges***

The *CPM Calculator* makes considerable use of naming ranges, which means grouping blocks of cells together under a label for easy referencing and intuitive use in subsequent modules. For clarity, it is recommended to highlight or frame these ranges in the spreadsheet. In the order of appearance, the ranges used in the *CPM Calculator* are “Input” for the three-column activity list filled in by the user, containing the names, durations, and successors of each activity, “Successors” for creating the predecessor matrix for additional clarity, which is the transposed of the successor matrix, “Col\_Allocation” for determining the column in an AON diagram in which the activity should be located, “Forward Pass” for looking up possible values during the forward pass, “Late\_Start” for the backward pass, and “Early\_Start” for calculating the free float.

## ***Input***

The schedule input is provided in the activity list by three columns of data, with one row for each activity to be included. The first column contains a brief name or identifying label for each activity. The second column defines the duration in a common time unit, in this case days, which must be consistent for all activities. The third column contains a list of all successor names for the activity in that row in a comma separated variable format. Using this activity list as the input format is the most common way in textbooks and scholarly publications to express project scheduling problems in their entirety. Lookup commands in subsequent modules in the spreadsheet require an alpha-numerically sorted data array to work correctly. However, it would have been counterproductive to limit the spreadsheet users to only such types of activity names. The *CPM Calculator* therefore uses an additional column placed at the front of the activity list to assign consecutive numbers to all activities and to keep the effort for the user at a minimum.

## ***Relationship Matrices***

To successfully execute the forward pass computations, the spreadsheet formulas must consider the dependency structure between all activities in the schedule. It is found that the ideal representation for this structure is a matrix. A column to the left of the matrix lists all possible predecessors as per the activity list and a row above the matrix lists all possible successors. Cells in the matrix therefore represent not an activity but the logic link between that pair of activities, which is indicated in binary form as “1” if existing and as “0” if no link exists. The successor matrix takes the form of an upper right triangle. The diagonal of the matrix from top left to bottom right is not filled, as activities are not connected to themselves. Interestingly, the equivalent predecessor matrix takes the form of a lower left triangle and is indeed the transposed of the successor matrix. The *CPM Calculator* therefore needs to only extract the information on relationships between activities once and through a simple array transposition applied to the matrix can create the new matrix as additional information to the user.

The spreadsheet command SEARCH is used to create the successor matrix, which finds a text string within another longer text string. If it is found, it returns the value of the first character of the search string within the target string; otherwise an error message is displayed. The character location is not used further, and the ISERROR command is used to convert possible error statements into a “0” as a negative binary statement. An outermost if statement converts the

positive value into a “1” to turn the matrix that is filled with this formula into a truth table of “1” if links are found and “0” if links are not found. The text to be searched for by column is fixed with the “\$” symbol. Cell references for columns and rows on the other hand are dynamic and will automatically be updated when the formula is dragged across and down the table to fill the entire matrix. Creatively using relative cell references allows the students to focus on the actual programming rather than typing command repeatedly.

### ***Column Assignment***

For drawing the schedule network on a grid where each activity has a specific column and row location, the column must be assigned first so activities are placed along the time axis in overall sequence and are connected by forward-pointing arrows only. The row allocation could use any permutation of activities in that column, but for clarity it should be selected such that it minimizes the number of crossings between logic links. It turns out that this is a mathematically extremely complex problem based in the specialty area of graph theory and will be addressed in future research (15). Note that the column assignment is an extra module in the *CPM Calculator* that is not necessary for subsequent modules to complete their computations. Equation 7 contains the recursive algorithm for the column assignment, which must follow the dependency structure.

$$C_j = 1 + \text{maximum (all } i) \{P_i\} \quad \text{Equation 7}$$

where  $C$  is the column assignment of an activity,  $P$  is the number of activities on a direct path from the overall start activity to activity  $i$ , and activity  $i$  is a direct predecessor to activity  $j$ . This column assignment is not clearly explained in most scheduling textbook, nor are tips for clear graph drawing provided. However, it has been documented in the literature on resource leveling of schedules via the minimum moment algorithm (8) and is commonly known as the sequence step (12).

The column assignment matrix in the spreadsheet again has the same setup as the successor matrix. Each of its columns contains the locations of all predecessors for a particular successor activity from among all the successor activities listed in a separate row above the matrix. The MAXIMUM command is used to pick the maximum column assignment of any predecessor and adds one, as the activity under consideration must be located one step further to the right in the diagram than its predecessors. In effect, the column assignment matrix accomplishes a count of all possible paths through the schedule network and picks the longest one that leads to the current activity. The HLOOKUP command is used to select the aforementioned maximum value and write it into said row above the matrix where the activity names are listed. Note that this recursive referencing does not violate the rule that circular references are not permissible in any computation, as they would lead to an unsolvable infinite loop that would abort the algorithm.

### ***Forward Pass***

The formula for the forward pass first checks its respective cell in the successor matrix for whether its value is “1” or “0” with the IF command, i.e., whether a link between the two activities indicated by its column and row position exists. The cells with a “1” in a particular row, i.e., all successors of an activity, must wait until that preceding activity has been completed

by reaching its EF time. The formula selects that EF time from the row in a separate column to the right of the forward pass matrix and places it into the forward pass matrix in the same place as the “1” values in the successor matrix. The successors, in turn, cannot commence before the last successors as listed in the rows underneath them has been completed by reaching their EF times, from which the formula chooses the maximum value with the MAXIMUM command and places it into a separate row above the forward pass matrix. Another column to the right of the forward pass matrix contains ES times, which are found by applying the HLOOKUP command to the aforementioned row, which is a horizontal array above the matrix. Adding the known duration from the input in the activity list to the ES time yields the EF time for each activity. The recursive process then continues until the maximum EF time of the overall finish activity is found as the duration of the entire project, also called makespan in manufacturing scheduling.

This seemingly circular set of recursive referencing by the formulas in fact is not a circular relationship and thus avoids an error message by the spreadsheet. The fact that a lookup formula is used rather than a direct formula circumvents this particularly challenging programming part.

### ***Backward Pass***

The formula for the backward pass uses the same concepts as for the forward pass, but in reverse. Note that it also is calculated in an upper right triangular matrix. The formula checks its respective cell in the successor matrix. It then selects the LF times for the activities listed in the row above the backward pass matrix from a separate column to the right of the matrix using the VLOOKUP command. More specifically, if the successor matrix contains a “1”, the formula looks up LS times and selects the minimum with the MINIMUM command. If the successor matrix contains a “0”, the formula looks up LF times and selects the maximum with the MAXIMUM command. While the forward pass matrix thus has identical values in a row (or no values, if no connection exists), the backward pass matrix is has identical values in a column (or no values, if no connection exists). Again, this somewhat confusing mechanism is not circular.

### ***Output***

The output range of the spreadsheet organizes the results that have been created in its various modules into a well-structured table. The criticality of each activity is evaluated by evaluating its total float and the text string “yes” or “no” is displayed depending on whether its TF is zero or not. The FF is always less than or equal to the TF and for critical activities thus also falls to zero.

### ***Scalability***

While the matrix structure of the spreadsheet can accommodate CPM schedules of arbitrary complexity, i.e. the entire upper right triangle in the successor matrix can theoretically be filled with connections for most interconnected and dense schedule possible, the spreadsheet has also been carefully set up to be fully scaleable. The number of activities determines the size of the square matrices in each module. The expansion to more than the initially permissible number of activities that a schedule can contain simply requires the user to add rows and columns in the core of these matrices until they reach the necessary size and pull the previously entered formulas into the new cells. Note that the formulas in Table 1 contain carefully defined “\$”

symbols that set selected column or row references fixed to allow for this expansion. All matrices must be aligned either horizontally or vertically for scalability. The authors found that a horizontal arrangement can simplify scaling to a mere three expansion points along both axes. The overall expansion is limited only by the available size of the spreadsheet itself, which under Microsoft® Excel is 256 columns and 65,536 rows, assuming that matrices could even be distributed onto different spreadsheets within the same workbook. More specifically, the forward pass and the backward pass each require one column per activity, thus halving the columns. Additionally, “overhead” space is necessary for various labels that structure the calculation.

## Object-Oriented Programming for Critical Path Method

### *Input and Object Initialization*

We have designed our object-oriented implementation to require the same parameters used by the spreadsheet approach. These parameters include the name, duration, and successors of each activity in form of an activity list. These values are read into the program from a tab-delimited text editor file, which can easily be produced from the spreadsheet and vice versa. At the onset of the programming code, objects are created for the activities. Note that in the following programming code the symbol “←” assigns a value to the variable it points at and the symbol “//” indicated non-executable commentary. Variable names are shown in italics.

### *Column Assignment*

```
//Propagation of Column IDs
for ColumnNumber ← 1 upto TotalNumberOfActivities
    for each Activity i (ascending) in Activities
        do if ColumnIDi = ColumnNumber
            then for each Activity j in Successorsi
                do if ColumnIDi > ColumnIDj
                    then ColumnIDj ← ColumnIDi + 1
```

In order to traverse the activities in the proper order when performing the forward pass and the backward pass of the CPM algorithm, the computer must first assign a *ColumnID* to each activity. This value is an indicator of the distance a particular activity lies from the start of the activity network graph. In the forward pass, activities need to be evaluated not in the order in which they appear in our input table, but rather in the order in which they appear in the graph. When each activity is created from the input, it is initialized with a *ColumnID* of 1. To assign the correct *ColumnID*, we must use a process comprised of three nested loops to propagate the appropriate *ColumnIDs* through the graph. The outermost loop simply increments a value called *ColumnNumber*, which is used by the inner loop. The inner loop iterates over all of the activities in the graph in the order that they appeared in the input table. For each activity *i*, a comparison is made between *ColumnID<sub>i</sub>* and the current *ColumnNumber* from the outer loop. If the two are equal, then the innermost loop is entered. This innermost loop iterates over the successors of the current activity. For each successor *j*, a comparison is made between *ColumnID<sub>i</sub>* and *ColumnID<sub>j</sub>*. If *ColumnID<sub>i</sub>* is greater than *ColumnID<sub>j</sub>*, then we set *ColumnID<sub>j</sub>* equal to *ColumnID<sub>i</sub>* plus 1. Upon the completion of the outermost loop, each activity will have the correct *ColumnID*.

### **Forward Pass**

```
//CPM Forward-Pass
for ColumnNumber ← 1 upto MaxColumnNumberactivities
  for each Activity i (ascending) in Activities
    do if ColumnIDi = ColumnNumber
      then if Predecessorsi = null
        then EarlyStarti ← 0
      else
        for each Activity h in Predecessorsi
          do if EarlyFinishh > EarlyStarti
            then EarlyStarti ← EarlyFinishh
        EarlyFinishi ← EarlyStarti + Durationi
```

In the forward pass, the computer assigns the proper values to *EarlyStart* and *EarlyFinish* for each activity in the graph. Similar to the column assignment process, the forward pass is completed through a series of nested loops and conditional statements. The outermost loop increments the *ColumnNumber*, allowing the computer to traverse the graph column by column. For each activity in the current column, the computer looks for any predecessors. If none exist, then we set the *EarlyStart* attribute for that activity to 0. In the likely case that the activity does have predecessors, the computer sets the *EarlyStart* attribute to the greatest *EarlyStart* of those predecessors. Once the computer has set the *EarlyStart* attribute for the current activity, it can set the *EarlyFinish* attribute by simply adding the *Duration* of the activity to its *Early Start* value. Once the outermost loop completes, the program can begin the backward pass.

### **Backward Pass**

```
//CPM Backward-Pass
for ColumnNumber ← MaxColumnNumberactivities downto 1
  for each Activity i (descending) in Activities
    do if ColumnIDi = ColumnNumber
      then if Successorsi = null
        then LateFinishi ← EarlyFinishi
      else
        for each Activity j in Successorsi
          do if LateStartj < LateFinishi
            then LateFinishi ← LateStartj
        LateStarti ← LateFinishi - Durationi
```

The backward pass is implemented in nearly identical fashion to that of the forward pass. In the backward pass however, the computer iterates over the columns of the graph in reverse. For each activity in the current column, the computer looks for successors instead of predecessors. If none exist, the *LateFinish* attribute for that activity is set to the same value as the *EarlyFinish* attribute. When the activity does have successors, the computer sets the *LateFinish* attribute to the least *LateStart* of those successors. Once the *LateFinish* has been set for an activity, the computer can set the *LateStart* attribute by simply subtracting the *Duration* of the activity from

its *LateFinish*. Upon the completion of the outermost loop in the backward pass, each activity in the graph will have been assigned a *EarlyStart*, *EarlyFinish*, *LateStart*, and *LateFinish* value from which other measures can be derived.

### ***Output***

The output of our object-oriented implementation is formatted identically to that of the spreadsheet approach so that the results of each can be easily verified by comparison to the results of the other. Our current object-oriented implementation does not save the output to a file, but instead displays it on the screen as a table. The contents of this table can be copied and pasted into spreadsheet software and many other applications as a means of saving the results.

### **Comparison of Spreadsheet with Object-Oriented Programming**

An OOP approach would use the appealing existing division into objects, the activities, which are related in a clearly defined sequence. It would follow a one-dimensional flow of individual commands including various loop statements to accommodate the case distinctions at forks in the dependency structure. While an object-oriented implementation of the CPM algorithm is rather efficient, both in terms of lines of code as well as required computational resources, it is pedagogically impractical unless the student has some significant prior experience with OOP and its related concepts. Without the proper background in OOP, such an implementation of the CPM algorithm might appear as a black box to the student, where the input enters and the output is released, but what happens inside would remain a mystery. Even with such experience, the code itself will only provide the student with a one-dimensional list of commands, and just a little insight into how those commands are executed iteratively and recursively to complete the CPM analysis. It is here where the spatially recursive spreadsheet approach offers a richer platform for learning to occur. In addition to clearly describing the steps in the algorithm, the spreadsheet provides a very useful snapshot of the data tables at each stage in the calculation, allowing the student to essentially view the inner workings of the black box. Instead of using the somewhat abstract concept of a loop to describe CPM iterations in OOP, the spreadsheet takes advantage of its more intuitive table structure by displaying the same formula in neighboring cells. Pointers to other cells are simply incremented or decremented by one. For example, if the formula in one particular cell is “=SUM(A4:D4)”, the equation in a neighboring cell may be “=SUM(A5:D5)”.

As discussed earlier, the scalability of the spreadsheet implementation is somewhat limited while the scalability of the object-oriented approach is arguably unlimited. For an actual commercial application, such a limitation on scalability would be quite a detriment. It must be recalled, however, that the spreadsheet is produced to aid in teaching CPM, not to provide an enterprise-level solution. With this in mind, it is reasonable to consider this limitation of the spreadsheet approach to be negligible. The true and perhaps most important distinction between the OOP approach and the spreadsheet approach is the way in which the student interacts with his or her implementation of the algorithm. In an object-oriented world, the development environment and the runtime environment are extremely different. While developing the software, the student uses an integrated development environment (IDE) or another text editor to write the code for a program while it is *not* running. Once enough of the code has been completed to run the program, the student will no longer be providing input and receiving output through the IDE or

text editor, but rather through the user interface that has been built into the program. While the program is running, the student cannot edit the code. The spreadsheet approach smoothes this dance between development environment and runtime environment by simply combining them. A beautiful feature of the spreadsheet is that the user can edit the software as it runs. Moreover, input is provided and output is received from the program using the exact same interface through which one implements the program. This is very appealing for educational purposes, as it allows the student to identify and fix mistakes –debug – the program while developing it. Also, the student will only need to be familiarized with a single piece of common commercial software instead of having to gain familiarity with the many development applications, techniques, and linguistic syntaxes that are involved in writing object-oriented code.

## **Conclusions and Future Research**

This paper has presented an innovative teaching tool called *CPM Calculator* that makes beneficial use of the two-dimensional nature of spreadsheets. Following a description of traditional teaching of CPM, a comparison with OOP has been made and the tool has been found to have several advantages from an educational point of view over both the classic textbook method of teaching CPM and an implementation with OOP that would be created by students.

Beyond the capabilities described in this paper, the tool already creates values for the independent float and interfering float as defined by Halpin and Woodhead (16). It also presents outputs in all three possible labeling conventions for counting start and finish dates, which are the start-of-day, the end-of-day, and the mixed convention. Currently ongoing research is adding capabilities in the Program Evaluation and Review Technique (PERT), which assumes probabilistic activity durations, and in Monte Carlo Analysis to the *CPM Calculator*. Due to the modular nature of the tool it is easy to exchange the duration column for a module that samples a random duration from a probabilistic distribution. It will therefore be possible to use the tool a test platform for planned research into schedule criticality indices and sub-criticality, where multiple runs of the analysis will be automatically tabulated. The tool will also support future research into measures of complexity of the network structure of the schedule under a new collaboration with experts in the mathematical area of graph theory on improving the graphical arrangement of the network by finding permutations of row assignments that would fulfill a triple objective function of obtaining short links with few bends and a low number of crossings.

Furthermore, different relationship types could be implemented, i.e., start-to-start, start-to-finish, finish-to-start, and finish-to-finish, each of them with their own lead or lag times (i.e., positive or negative durations to pass between the activity end points that they connect), albeit with a significantly increased difficulty in the matrix computations. Finally, true graphical capabilities that can be controlled by the student could be added using programming code in a macro language, whose syntax and commands would have to be included in the course materials.

## Bibliography

1. Galloway, P. D. (2006). "Comparative Study of University Courses on Critical-Path Method Scheduling." *Journal of Construction Engineering and Management* 132(7): 712-722.
2. Weaver, P. (2006). "A Brief History of Scheduling: Back to the Future." Conference paper presented at myPrimavera06, April 4-6, 2006, Hyatt, Canberra, Australia, available at: <[http://www.pmforum.org/library/papers/2006/A\\_Brief\\_History\\_of\\_Scheduling.pdf](http://www.pmforum.org/library/papers/2006/A_Brief_History_of_Scheduling.pdf)>, accessed January 10, 2007.
3. Davis, E. W. (1974). "CPM Use in Top 400 Construction Firms." *Journal of the Construction Division, Proceedings of the American Society of Civil Engineers* 100(CO1): 39-49.
4. Tavakoli, A., Riachi, R. (1990). "CPM Use in ENR Top 400 Contractors." *Journal of Management in Engineering* 6(3): 282-295.
5. Kelleher, A. H. (2004). "An Investigation of the Expanding Role of the Critical Path Method by ENR's Top 400 Contractors." Thesis submitted to the Faculty of Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of Master of Science in Civil Engineering, Virginia Polytechnic Institute and State University, Blacksburg, VA.
6. de la Garza, J. M., Kelleher, A. H. (2005). "An Investigation of the Expanding Role of the Critical Path Method by ENR's Top 400 Contractors." *Presentation, 2<sup>nd</sup> Annual PMI College of Scheduling Conference, Scottsdale, AZ, May 22-25, 2005*, Project Management Institute, Newtown Square, PA, available at <<http://www.pmicos.org/topics/topic%20-%202005-09.pdf#search=%22pmicos%20kelleher%22>> [accessed September 25, 2006].
7. Galloway, P. D. (2006). "Survey of the Construction Industry Relative to the Use of CPM Scheduling for Construction Projects." *Journal of Construction Engineering and Management* 132(7): 697-711.
8. Callahan, M. T., Quackenbush, D. G., Rowings, J. E. (1992). *Construction project scheduling*. Irvin / McGraw-Hill, Boston, MA.
9. Patrick, W. C. (2004). *Construction project planning and scheduling*. Pearson Education / Prentice Hall, Upper Saddle River, NJ.
10. Mubarak, S. (2005). *Construction project scheduling and control*. Pearson Education / Prentice Hall, Upper Saddle River, NJ.
11. Feigenbaum, L. H. (2002). *Construction scheduling with Primavera Project Planner®*. 2<sup>nd</sup> ed., Pearson Education / Prentice Hall, Upper Saddle River, NJ.
12. Hegazy, T. (2002). *Computer-based construction project management*. Pearson Education / Prentice Hall, Upper Saddle River, NJ.
13. Buttlerwerth, J. W. (2005). *Computer integrated construction project scheduling*. Pearson Education / Prentice Hall, Upper Saddle River, NJ.
14. Lucko, G. (2005). "Reviving a mechanistic view of CPM scheduling in the age of information technology." *Proceedings of the 2005 Winter Simulation Conference*, Orlando, Florida, December 4-7, 2005, Institute of Electrical and Electronics Engineers, Piscataway, New Jersey, 1533-1540.
15. Lucko, G. (2006). "An Activity and Arrow Arranging Algorithm for Clarity in Schedule Network Diagrams." *Proceedings of the 2006 International Conference on Computing in Civil Engineering of ASCE at the Joint International Conference on Computing and Decision Making in Civil and Building Engineering*, Montréal, Québec, Canada, June 14-16, 2006, American Society of Civil Engineers, Reston, Virginia.
16. Halpin, D. W., Woodhead, R. W. (1998). *Construction management*. 2<sup>nd</sup> ed., John Wiley & Sons, New York, NY.