

A COMPUTATIONAL METHOD FOR CALCULATING UNCERTAINTY IN PROJECT NETWORKS WITH DEPENDENCIES

Santiago Erquicia, University of Minnesota Duluth

Abstract

Monte Carlo simulation is the most frequently used method for quantifying project uncertainty, but getting accurate results for large project networks requires large computer resources. A computational method is presented in this paper that accurately computes the uncertainties in task times and project duration while requiring less computational time than simulation. The technique utilizes series and parallel operators for combining independent uncertainties in a project network. The method applies the operators in the proper sequence to preserve independence, and an extension to the methodology that solves those cases where the condition of independence cannot be met is also presented.

Introduction

The most widely used method to calculate the project ending time uncertainty is PERT/CPM. PERT assumes that every activity is beta distributed with a predefined shape. It also assumes the applicability of the central limit theorem, so the results are all normally distributed. Both assumptions are not totally correct [Archibald and Villoria 1967]. For calculation purposes it uses the expected values and then adds the variability. The normal arithmetic for certain numbers applied to the expected values doesn't give the same results as the probability theory and it doesn't take into account the accumulated probabilistic dependencies between calculated results. The main advantage of this method is that it is quick to calculate and it is included in most of software packages.

Monte Carlo simulation is the recognized method to get accurate results. The method doesn't impose any restriction to the probability distribution of the task's durations and it is also network agnostic. The only drawback is that it is computer resource consuming.

Proposed Method

The proposed method of this paper addresses the weaknesses of PERT/CPM. It provides an exact result based on probability theory, while avoiding the high-resource requirements of Monte Carlo simulation. It uses parallel and a series operator combined with a network-solving technique.

The only restriction on the network at this moment, mainly because of lack of research, is that the only allowed relationship is finish-to-start. There are other types of relationships that are common in PERT/CPM and Monte Carlo but most of the real

networks can be specified without them [Goodpasture 2004, p. 191].

Even though it is possible to calculate the slack times of each task, by comparing the task duration against the network, the algorithm presented here has not been expanded to that case. It will be addressed in future research. Despite slack times are not calculated, it would not be difficult to see the "pure slack" of a task or the duration that a task is 100% non critical. In that case, the task can be moved without having to recalculate the whole network.

Parallel and series operators. The parallel and series operators calculate the resulting duration of two durations in series or parallel based on probability theory. Two tasks, so its durations, are in series when there is a direct finish-to-start relationship between them. They are in parallel when both have the same direct successor.

The operators are based on probability theory and use discrete random variables with a certain precession to represent the continuous random variables.

The series operator calculates the resulting times and their associated probability by computing all the possible combinations of the uncertain durations, adding the time and assigning its probability as the multiplication of the individual contributions.

The parallel operation is similar to the series operation but instead of adding the time values, it assigns only the maximum value.

Both operators assume that each uncertain duration used as parameter is independent of each other. Although this is an important restriction, many network configurations allow simplifying them up to the point where this constraint is valid. Even in the cases were this situation cannot be reached, the same operations can be used for part of the network.

Calculation of starting and ending times. The ending time of a task is calculated using the series operator, represented by "add", using as parameters the starting time of the task and its duration as shown in Equation (1).

$$T_i^E = \text{add}(T_i^S, T_i^D) \quad (1)$$

where:

$$\begin{aligned} T_i^E &= \text{Task}_i \text{ ending time} \\ T_i^S &= \text{Task}_i \text{ starting time} \\ T_i^D &= \text{Task}_i \text{ duration} \end{aligned}$$

The starting time of a task never depends on the duration of that task, so they are independent variables and the independence condition of the operator is fulfilled.

In order to calculate the starting time of a certain task, there are different alternatives. The most basic one (brute force method) is to do it by solving the network between the project starting time and the tasks that have as a successor the given task. This method can always be done and it does not have any other restriction.

In order to avoid making the same calculations over and over again using the previous method and to make the calculation process quicker there are different scenarios that should be analyzed.

If the task has only one predecessor, that predecessor and the task are in series. In this case the ending time of the predecessor is the same as the task's starting time as shown in Equation (2).

$$T_j^S = T_i^E \quad (2)$$

where:

j, i such Task_i is predecessor of Task_j

If the task has more than one predecessor and they don't have any common ancestor, their ending times are going to be independent. Therefore, there is no inconvenience in taking those immediate predecessors' ending time and apply the parallel operator, represented by "max", to get the starting time of the task in question as shown in Equation (3).

$$T_j^S = \max(T_i^E) \quad (3)$$

If any of these shortcuts cannot be applied, the task's starting time should be calculated by applying the parallel operation for all the possible paths from the project's starting time to the task's predecessors. In order to use the previously mentioned operators, the network should be solved in a certain way to avoid the dependency assumption of the operators.

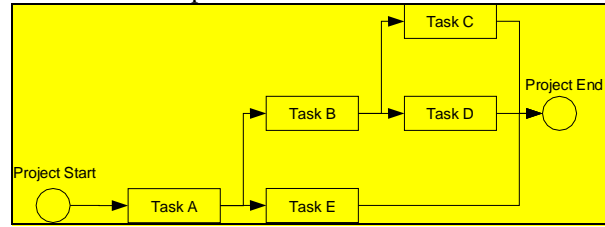
Network simplification. The objective of the network simplification is to provide the order and parameters to use with those two operators while avoiding as many as possible duplicated tasks in order to satisfy the restrictions imposed by those operators.

The only operation that can be simplified is the parallel operation. It is the only operation that can have a duplicated partial parameter that can be factored out by putting it in series with a parallel operation of the branches without that common part.

In order to be able to make a recursive algorithm, the project starting time has to be created as a task with zero duration or milestone (task₀).

Exhibit 1 shows the network used as an example to show the proposed method.

Exhibit 1. Example Network



Direct and indirect successors matrix. The direct successors matrix (M_D) is a representation of the direct relationships between the different tasks in the network. It is a squared matrix where the number of columns and rows are equal to the number of tasks in the network, including the project start and end milestones. Each position (i,j) has a value of one if task_i has as a successor task_j.

The direct/indirect successor matrix (M_I) shows if a given task is a successor of another task either by itself or by one of its predecessors. It is a square matrix of the same size of the direct matrix. Each position (i,j) has a value equal or greater than one if task_j has a successor relationship with task_i.

Equation (4) provides a simple method to compute the direct/indirect relationship matrix based on the direct relationship matrix.

$$M_I = (I - M_D)^{-1} \quad (4)$$

where:

I = Identity matrix

The direct and indirect matrixes of the example are shown in Exhibit 2 and Exhibit 3 respectively. For the creation of those matrixes it was considered that the project's start and end milestones are the first and second task respectively and the other tasks are sorted alphabetically.

Exhibit 2. Direct Relationship Matrix

To \ From	Start	End	Task A	Task B	Task C	Task D	Task E
Start	0	0	1	0	0	0	0
End	0	0	0	0	0	0	0
Task A	0	0	0	1	0	0	1
Task B	0	0	0	0	1	1	0
Task C	0	1	0	0	0	0	0
Task D	0	1	0	0	0	0	0
Task E	0	1	0	0	0	0	0

Exhibit 3. Direct/Indirect Relationship Matrix

To \ From	Start	End	Task A	Task B	Task C	Task D	Task E
Start	1	3	1	1	1	1	1
End	0	1	0	0	0	0	0
Task A	0	3	1	1	1	1	1
Task B	0	2	0	1	1	1	0
Task C	0	1	0	0	1	0	0
Task D	0	1	0	0	0	1	0
Task E	0	1	0	0	0	0	1

Each value in the direct/indirect matrix represents the different possible paths to go from task_i to task_j given the relationships.

Creation of the path list. The process creates a list with all the different possible paths from the project's starting task to every task's predecessor based on the relationships between the different tasks.

The algorithm looks at every task's successor that has task_i as a direct or indirect successor to determine if that successor has to be processed. The algorithm uses the indirect dependencies matrix for each task to avoid analyzing the entire network and see at the last moment that a branch should not be included. This process is used for every task and all of its successors in a recursive form.

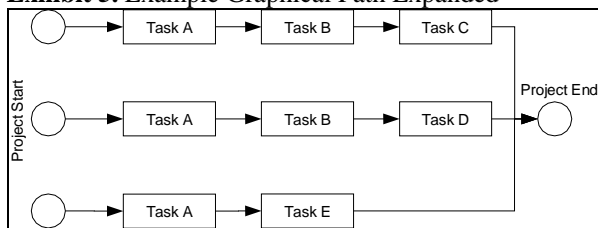
The process starts using the project's starting task (task₀). If the task being analyzed has task_i as direct successor, it returns a list with only the current task. Otherwise, it is analyzed which successor contains task_i as an indirect successor and this same process is executed starting at that successor. Every path that leads from any of those successors to task_i are added to the list of paths. Finally, the current task is added to every element in the list.

Exhibit 4. Example Path List

```
[
  Project Start , Task A , Task B , Task C
  Project Start , Task A , Task B , Task D
  Project Start , Task A , Task E
]
```

Exhibit 4 shows the list of possible paths by applying this algorithm to find the project ending time for this particular example. Exhibit 5 shows the graphical representation of the list.

Exhibit 5. Example Graphical Path Expanded



Creation of the initial arithmetic tree. An arithmetic tree is a hierarchical organization of the different operators and its parameters. It describes which operators are going to be used, in which order, and with what parameters. The hierarchical structure provides an organizational method easy to create and manipulate with a computer program. A common nomenclature used in tree manipulation is to call the elements at the next level of the hierarchy, that have a direct relationship, “children”. We use “grandchildren” to identify elements two levels lower than its “grandparent”. Also, a “subtree” is a part of a tree.

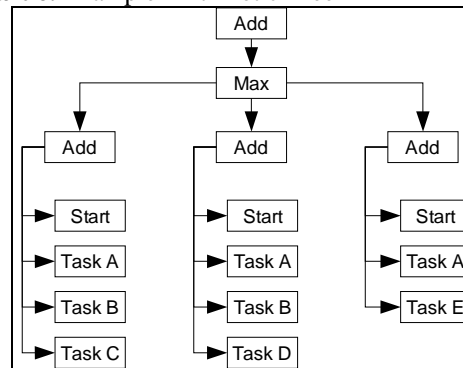
An initial tree with the required different operations is created based on the previously obtained path list. The path list can contain one or more possible paths. The first operation of the tree is always a series operation. If there is only one possible path, the result is only a series operation with all the tasks involved as children. Otherwise, the first-operation's child is a parallel operation with as many series operations children as different possible paths. This tree structure assures that every parallel operation has series operations as direct children. It can happen that a series operation only has one child.

The result of the initial arithmetic tree in the example, for finding the project ending time, is:

```
add(max(add(Start, Task A, Task B, Task C),
  add(Start, Task A, Task B, Task D),
  add(Start, Task A, Task E)))
```

A graphic representation of this arithmetic tree can be seen in Exhibit 6.

Exhibit 6. Example Arithmetic Tree



For future use, a description of part of a tree is the textual representation of that particular part of the tree. In the example presented here, a description could be “Task A”, “add(Task A, Task B, Task D)”, or “max(add(Start, Task A, Task B, Task C), add(Start, Task A, Task B, Task D), add(Start, Task A, Task E))”. These descriptions are going to be used as a method to see if two or more branches are equal.

Simplification of the arithmetic tree. The simplification procedure starts with the initial arithmetic tree at its first operation. The parallel operations can be simplified, the series operation instead cannot and it should be analyzed if its child operations contain anything that can be simplified.

Positioned in the tree at a parallel operation, it is analyzed how many times a given grandchild description is repeated. We have to look at the grandchildren because each parallel operation has an obligatory series operation as children and what it is important is these series operations' children.

Every grandchild's element that it is repeated as many times as existing branches is moved as a child of the parent series operation. The parent series operation must have only one of all those grandchildren. The operation means that if the same subtree is founded in every path, it can be factored out and put in series with the rest of the tree.

Using the previous example, the first parallel operation grandchildren are Start, Task A, Task B, Task C, Task D, and Task E. The number of appearances of each of those grandchildren is 3, 3, 2, 1, 1, and 1 respectively. The number of branches is 3, so the only grandchildren that can be factored out is Start and Task A. The result of this operation applied to the example is:

add(Start, Task A, max(add(Task B, Task C), add(Task B, Task D), add(Task E)))

After factoring out all the elements that belonged to every path, the second most repeated element (more than one time) is factored out. If there are two or more elements with the same number of repetitions, any of those elements can be selected. This element cannot be moved back to the parent of the element we are processing because it is not included in every branch. A new series operation is created and inserted as a child of the parallel operation we are processing. The selected element is moved as a child of the new series operation. A new parallel operation is created and set as a child of the new series operation. The trees that were a child of the parallel operation, eliminating the factored element, is inserted as children of a new parallel operation.

Continuing with the example, Task B is the most repeated grandchild and should be processed. The result of this operation in the example is:

add(Start,Task A, max(add(Task B,max(add(Task C), add(Task D))), add(Task E)))

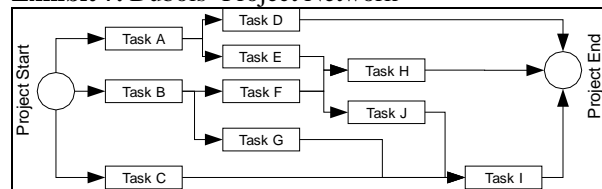
Because at this point the tree changed substantially, the simplification process is executed again over the parallel operation we are processing and then over its parent. In our example, there is no further possible simplification because each task appears only once.

Finding the solution. The result can be easily found if there is no repeated task at the end of the simplification process. If that is the case, starting at the first operation, we see if all the operation's parameters are already calculated. "Calculated" means that they are tasks or trees already solved. If they are all ready, we just apply the operations over them. If they are not, we solve each of those parameters in the same way.

If there is at least one task repeated in the tree, it is not going to be possible to use the operators because at one moment two durations are not going to be

independent. This duplication is not a problem if the durations are known (not uncertain). The proposed solution is to solve the network replacing those repeated tasks with all the possible duration combinations. Once each uncertain duplicated task is replaced by a non-uncertain duration, there is no problem in using the operators with those duplicated durations. The solution founded with each of these combinations is multiplied by the total probability of occurrence of the combination and, finally, all the resulting probability density functions (PDF) are added. The problem with the proposed solution is that the time required to perform it grows exponentially with the possible durations of each task and the number of repeated tasks.

Exhibit 7. Dubois' Project Network



Example calculation. The network used to compare the results of the different methods is shown in Exhibit 7 (Dubois, Didier, and Prade 1980). Each task duration was set as a triangular distribution with the parameters shown in Exhibit 8. The reason for using this network is that it provides a good set of different task relationship's conditions and has been used in other researches [Nasution 1994].

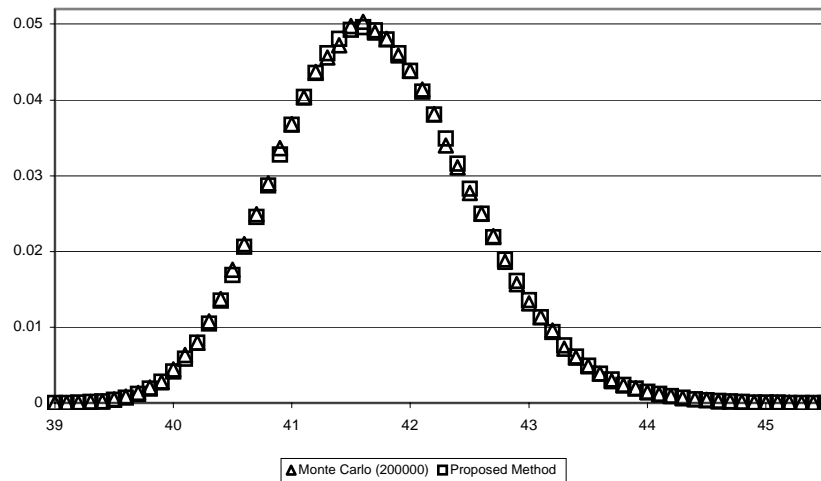
Exhibit 8. Dubois Time Duration Estimates

Task	Time		
	Minimum	Most Likely	Maximum
A	9	10	12
B	9	10	12
C	29	30	32
D	29	30	32
E	9	10	12
F	9	10	12
G	19	20	22
H	18	19	21
I	9	10	12
J	8	9	11

The calculation of the project ending time in this network is impossible to do without implementing the conditional solution method mentioned before. The result of the arithmetic tree simplification is:

add(max(add(Task I, Task C), add(max(add(Task I, Task G), add(max(add(Task H), add(Task I, Task J)), Task F))), Task B)), Task I)

Exhibit 9. Dubois Project Ending Time



```
add(max(add(Task D),add(Task H,Task E)),  
Task A)), End, Start)
```

In this example, it is impossible to eliminate the duplications of Task I and H in order to solve the project ending time.

The project's ending time probabilities are shown in Exhibit 9. The proposed method is plotted with the results of doing the Monte Carlo simulation with 200,000 iterations showing the highly accurate result of the proposed method.

The simulation of the network was done in custom-made programs using a time precision of 0.1 for both the histogram creation in Monte Carlo and the tasks duration's PDF for the proposed method. The time requirement to solve the network with the new method was 8.4 seconds while for the Monte Carlo simulation was 72.4 seconds. It is important to know that those time requirements can only be used as relative values given that the programs were not written in a language that provides the quickest execution, like C. The numeric time values can be used just for the purpose of comparison. The program that implemented the proposed method does not contain any of the possible optimizations mentioned here and it just solves all the tasks starting time using the "brute force" method.

The conditional solution of this problem, given the 0.1 precision used to create the duration's PDF, implied to solve the network 961 times. If that were not the case, the required solving time would be drastically decreased.

Evaluation of the Method

The proposed method was evaluated for correctness and accuracy against Monte Carlo simulations by generating 20 random networks each having 30 tasks.

All the networks were simulated with 2,000, 20,000, and 50,000 iterations when using Monte Carlo and the comparisons were made with the project ending time. Exhibit 10 shows the total differences in the probability at each time between the proposed method and the Monte Carlo simulations for every random network tested.

The correctness of the proposed method is showed by the near zero Cumulative Difference values for every network test simulated. These results do not show any bias in the calculation method and any difference between both methods is almost totally compensated.

Looking at the Absolute Cumulative Difference values shows that the proposed method is accurate. As the number of iterations used with Monte Carlo grows, the sum of the absolute differences gets smaller. It would be reasonable to assume that the more iterations used in Monte Carlo, the closer the results would be to the one obtained with the proposed method.

A graphical representation of the accuracy is shown in Exhibit 11, showing the different results from Monte Carlo and the proposed method using the Dubois Network.

Conclusions

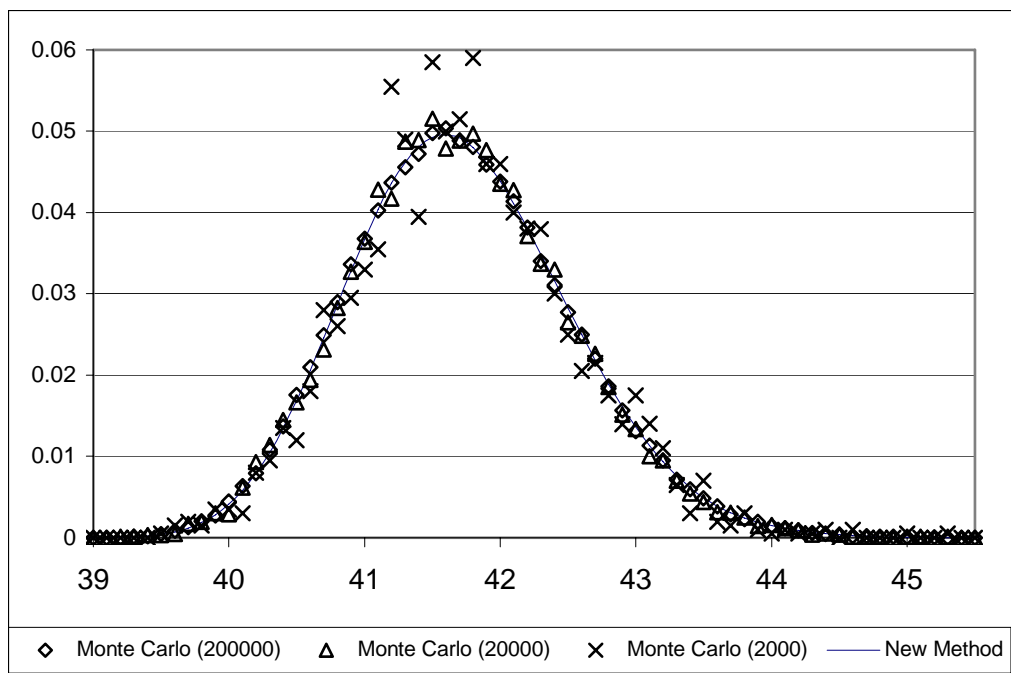
The proposed method correctly computes the uncertainty of every task starting and ending time, including the project ending time. The proposed method provides more accurate results compared to Monte Carlo simulation when doing it with a reasonable number of iterations. As the number of iterations used in Monte Carlo increases, its results approach the results of the proposed method.

An advantage of the proposed method over Monte Carlo is that it is not required to simulate the entire network if something is changed or added. It can only

Exhibit 10. Evaluation Test Result's Difference

Test	Cumulative Difference			Absolute Cumulative Difference		
	Monte Carlo 2000	Monte Carlo 20000	Monte Carlo 50000	Monte Carlo 2000	Monte Carlo 20000	Monte Carlo 50000
1	4.0657E-13	4.0655E-13	4.0655E-13	0.19539	0.06340	0.03836
2	-1.2784E-13	-1.2783E-13	-1.2783E-13	0.05662	0.20050	0.04366
3	3.9930E-13	3.9928E-13	3.9929E-13	0.17780	0.06568	0.04086
4	-1.2703E-13	-1.2703E-13	-1.2703E-13	0.18596	0.06203	0.18596
5	-2.2421E-13	-2.2423E-13	-2.2423E-13	0.18389	0.05064	0.03441
6	2.2378E-13	2.2378E-13	2.2378E-13	0.16607	0.05786	0.03492
7	-1.0938E-13	-1.0940E-13	-1.0940E-13	0.17556	0.05986	0.03706
8	3.8431E-13	3.8431E-13	3.8431E-13	0.19881	0.06394	0.04070
9	4.0718E-13	4.0718E-13	4.0719E-13	0.21402	0.05934	0.04062
10	4.0728E-13	4.0728E-13	4.0727E-13	0.19994	0.05424	0.03365
11	1.5671E-13	1.5669E-13	1.5671E-13	0.18583	0.05625	0.03684
12	2.5687E-13	2.5687E-13	2.5687E-13	0.17713	0.05973	0.03562
13	3.9528E-13	3.9528E-13	3.9528E-13	0.20284	0.06627	0.03958
14	-2.1941E-13	-2.1942E-13	-2.1942E-13	0.20845	0.07127	0.04379
15	1.5769E-13	1.5769E-13	1.5770E-13	0.18869	0.05894	0.03391
16	1.4908E-13	1.4908E-13	1.4907E-13	0.18827	0.05218	0.03550
17	-7.9521E-14	-7.9534E-14	-7.9526E-14	0.19987	0.05134	0.03390
18	2.5936E-13	2.5935E-13	2.5936E-13	0.21182	0.05650	0.03686
19	-2.1061E-13	-2.1061E-13	-2.1061E-13	0.18662	0.07187	0.04087
20	-8.8355E-14	-8.8366E-14	-8.8376E-14	0.17239	0.05306	0.03462

Exhibit 11. Accuracy Plot Using Dubois Network



be calculated the part of the network affected. Monte Carlo would require an entire simulation again even though the task was added at the end of the network.

The proposed solution has the problem that its time requirements grows exponentially, for those networks that cannot be solved without duplicating tasks, with the precision used and number of duplicated tasks. Although many real project networks can be solved without using the conditional solver, or at least with few duplicated tasks, the possibility exists. The problem can easily appear if you want to add resource constraints to the project network.

Future Work

Future research would be necessary to find a solution to the problem founded when it is required a conditional solution. It will be required to find some algorithm to minimize the number of conditional solutions required to get a "good" result.

It would be also important to incorporate other task relationships other than finish-to-start. This requirement is not critical given that most of the project networks only use this type of relationship.

Although during the research, slack times and criticality were calculated by hand, it was not possible to further analyze a way to do it automatically in every case. Further research would be required to develop this methodology using a similar approach to the one used in this paper.

Finally, the method should be expanded to consider resource constraints.

Acknowledgments

This research was generously supported by a grant from the Northland Advanced Transportation Systems Research Laboratory, a cooperative research and education initiative of the Minnesota Department of Transportation, the University of Minnesota Center for Transportation Studies and its Intelligent Transportation Systems Institute, and the College of Science and Engineering at the University of Minnesota Duluth.

References

- Archibald, Russel D., and Villoria, Richard L., *Network-based Management Systems (PERT/CPM)*, Wiley (1967)
- Dubois, Didier J. and Prade, Henri, *Fuzzy Sets and Systems: Theory and Applications*, Academic Press, New York (1980).
- Goodpasture, John C., *Quantitative Methods in Project Management*, J. Ross Publishing (2004).
- Nasution, Sofjan H., "Fuzzy Critical Path Method", *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 24, No. 1, (January 1994), pp. 48-57.

About the Author

Santiago Erquicia is a MS in Engineering Management student at the University of Minnesota Duluth. He received a Bachelor of Chemical Engineering from the Instituto Tecnológico de Buenos Aires, Argentina, in 1999 and has 4 years of experience as an operations business analyst and project manager at Transportadora de Gas del Norte, Argentina.