[Mn/DOT / CTS Report Template]

[This template is offered as an example. Please refer to the ORS website for the latest guidelines on submitting reports for publication. Directions and information in brackets should be removed before final submission.]

[Front cover: here.
Front cover will be created by Mn/DOT and will contain official Report Number and logos of sponsoring agencies.]

Technical Report Documentation Page

1. Report No.	2.	3. Recipients Accession No.
4. Title and Subtitle Quantification of Uncertainty in Transportation Infrastructure Projects		5. Report Date 6.
7. Author(s) Ryan G. Rosandich, Ph.D. and Santiago Erquicia		8. Performing Organization Report No.
9. Performing Organization Name and Address Department of Mechanical and Industrial Engineering University of Minnesota Duluth 105 Voss-Kovach Hall, 1305 Ordean Court Duluth, MN 55812		10. Project/Task/Work Unit No. 11. Contract (C) or Grant (G) No.
12. Sponsoring Organization Name and Address Minnesota Department of Transportation 395 John Ireland Boulevard Mail Stop 330 St. Paul, Minnesota 55155		13. Type of Report and Period Covered 14. Sponsoring Agency Code
15. Supplementary Notes		

16. Abstract (Limit: 200 words)

Monte Carlo simulation is the currently accepted method for quantifying uncertainty in projects. It was the goal of the research presented in this report to develop a purely computational technique, based on traditional probability theory, for quantifying project uncertainty with accuracy equal to or greater than that of Monte Carlo simulation. Series and parallel operators were developed for combing independent task uncertainties in project networks. The operators were used to compute overall project uncertainty given individual task uncertainty, and to calculate slack and the degree of criticality for each task. Additional techniques were developed to deal with networks where the series and parallel operator were not enough, specifically those with path dependencies. Results equal or exceed the accuracy of Monte Carlo simulation, but computational times exceed those of Monte Carlo simulation for networks with many dependencies.

17. Document Analysis/Descriptors		18. Availability Statement No restrictions. Document available from: National Technical Information Services, Springfield, Virginia 22161	
19. Security Class (this report) Classified	20. Security Class (this page) Classified	21. No. of Pages	22. Price

Quantification of Uncertainty in Transportation Infrastructure Projects

Final Report

Ryan G. Rosandich, Ph. D.
Associate Professor
Department of Mechanical and Industrial Engineering
University of Minnesota Duluth

Santiago Erquicia
Engineering Management Graduate Student
Department of Mechanical and Industrial Engineering
University of Minnesota Duluth

July 2004

Published by
Minnesota Department of Transportation
Office of Research Services
Mail Stop 330
395 John Ireland Boulevard
St. Paul, Minnesota 55155-1899

Acknowledgements

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.

Table of Contents

List of Tables

List of Figures

Executive Summary

Dealing with the uncertainty inherent in future events is a challenging aspect of project management. A project manager's job is to plan and carry out a complex sequence of tasks, and to do so on schedule and within budget. The schedule and budget are based on estimates of how long tasks will take to accomplish and how much various materials will cost, and include a great deal of uncertainty. In addition to those uncertainties, project managers face external or environmental causes of uncertainty such as limited resources, unpredictable weather, and variable funding, to name a few. The management of project uncertainty is further complicated by the fact that the risks are not independent. A variation in the schedule, for example, will often have an effect on resources and cost.

A popular approach to the quantification of uncertainty in projects utilizes Monte-Carlo simulation. With this approach the expected values of duration and cost for each task are replaced with probability distributions that reflect the uncertainty in those estimates, and the project is executed repeatedly in simulated time to determine the effect of the task uncertainties on project outcomes.

Simulation-based methods are widely used in project planning, and most risk management software packages rely on Monte-Carlo simulation. Simulations can involve thousands of iterations of thousands of calculations, however, so they are complex and time consuming to perform. For this reason they are used primarily in the planning and decision making stages of a project, and not during actual project execution. It is desirable to develop a method of calculating project uncertainty that is computationally simpler, so it could be integrated with project management software and used in all phases of project management.

Given probability distributions that represent uncertain task durations, for example, it is desirable to combine those individual uncertain durations in some way to compute the overall project duration, and the uncertainty therein. Two operators, a series operator and a parallel operator, were developed to help accomplish this. A procedure was developed that applies the series and parallel operators repeatedly to a network to compute the overall project duration and uncertainty.

The computation of uncertainty in task slack times was also addressed. In an uncertain network the computation of slack becomes very complex. There may be many possible critical paths in an uncertain network, each with some probability of being critical. In order to calculate the slack for a particular task, its uncertain duration must be compared with all other possible parallel paths through the network. To accomplish this, one more operator, the *negate* operator, was required. This operator essentially reverses the time direction of a task duration and its uncertainty, so uncertain durations can be 'subtracted' in series as well as added.

Because slack is the difference between two uncertain task durations it can take on both positive and negative values. Tasks with completely negative slack are always critical, and tasks with completely positive slack are never critical. Tasks with slack on both sides of zero have some probability of being critical, and some probability of being non-critical. The area beneath the negative portion of the slack curve is defined as *criticality*, the probability of the task being critical.

A major limitation of the computational method is the fact that it requires that the project network be completely simplified using the series and parallel operators. Although this is possible for many practical networks, there are also many networks for which simplification is not possible due to path dependencies.

An enhancement to the computational method that deals with path dependencies was developed. The enhancement involves replicating tasks that participate in more than one path, and handling the durations of the replica tasks as if they were fixed. This avoids the dependencies, but introduces a great deal of additional computation when two or more tasks must be replicated.

The original and enhanced computational techniques were tested against Monte Carlo simulation. The methods were tested on many networks, and used to calculate many uncertain quantities including task start times, task end times, task slack and criticality, and overall project duration. The results of these computations were compared to the simulation results for the same networks.

The only practical limitation on the computational methods presented is the fact that once the replication of tasks becomes necessary, the computation time required becomes exponential in the number of replicated tasks. For the example, the computation time with no replicated tasks is about a second, with one replicated task a few seconds, and with two replicated tasks somewhat over a minute. Further investigation demonstrated that networks requiring three tasks to be replicated were computed in about 45 minutes, and if four tasks were to be replicated the computation would take on the order of a day, and five replicated tasks would take on the order of a month to compute. In comparison, very good results can be achieved using Monte Carlo simulation with about 200,000 iterations, and this requires a few minutes of computing time on the networks investigated in this research. It is likely that a hybrid technique that uses the serial and parallel operators to simplify the project network as much as possible, and then employs Monte Carlo simulation on the simplified network would be the most efficient approach to computing overall project duration uncertainty.

The conclusions based on the evaluation of the computational methodology presented in this report can be summarized as follows:

- The method accurately computes the effect of uncertainties in individual task durations on the overall project duration in project networks.
- The method accurately computes the uncertainty distributions in task slack times, and the resulting task criticality.
- Task criticality is in fact the area beneath the negative portion of the slack time distribution.
- Slack time distributions for tasks in series are identical.
- Criticality divides among parallel paths
- The total criticality for the project, as computed at the Start and End nodes, is 100%.
- The method requires exponential computational time on project networks with path dependencies, but can be used to partially simplify those networks to facilitate more rapid Monte Carlo simulation.

Chapter 1
Introduction

There are no facts about the future. This simple statement seems obvious, yet engineers and managers are always trying to predict or manipulate the future in some way. In a typical engineering management program, for example, courses are taught with topics like forecasting, operations management, and project management in which predicting, planning, and even controlling future events are discussed.

Dealing with the uncertainty inherent in future events is especially challenging in the field of project management. A project manager's job is to plan and carry out a complex sequence of tasks, and to do so on schedule and within budget. The schedule and budget are based on estimates of how long tasks will take to accomplish and how much various materials will cost, and include a great deal of uncertainty.

In addition to those uncertainties, project managers face external or environmental causes of uncertainty such as limited resources, unpredictable weather, and variable funding, to name a few. The management of project uncertainty is further complicated by the fact that the risks are not independent. A variation in the schedule, for example, will often have an effect on resources and cost.

The management of the unknowns and/or uncertainties involved with project management is known as *project risk management* (PRM), and it is currently one of the main areas of interest in the project management community [1]. The Project Management Institute (PMI), the largest professional organization in the world dedicated to project management, has identified risk management as one of the eight main areas of the Project Management Body of Knowledge [2], and one of the most active PMI Specific Interest Groups (SIG) is the one dedicated to risk management. The Risk Management SIG lists 108 risk management software tools on its website [3], and a recent survey of research on risk management [4] identified 241 references.

Risk management experts Chapman and Ward have recently argued that the term 'risk' should be replaced with 'uncertainty' in project management [5]. They argue that risk management is too narrowly focused on events, and that risk always has a negative connotation. They contend that replacing PRM with the broader project uncertainty management (PUM) will allow project managers to better understand the sources of uncertainty in projects, and to take advantage of the opportunities related to uncertainty as well as dealing with the threats.

There is great interest in improving PUM by quantifying the effect of uncertainty on project outcomes. Projects could be managed more effectively if knowledge of uncertainty could be incorporated into the project planning, monitoring, and control processes.

A popular approach to the quantification of uncertainty utilizes Monte-Carlo simulation. With this approach the expected values of duration and cost for each task are replaced with probability distributions that reflect the uncertainty in those estimates, and the project is executed repeatedly in simulated time to determine the effect of the task uncertainties on project outcomes.

After many simulations probability distributions can be determined for outcomes like total project cost and project duration. Those distributions reflect the level of uncertainty in the entire project, and represent a far more realistic predictor of actual project outcomes.

Simulation-based methods are widely used in project planning, and most risk management software packages rely on Monte-Carlo simulation. Simulations can involve thousands of iterations of thousands of calculations, however, so they are complex and time consuming to perform. For this reason they are used primarily in the planning and decision making stages of a project, and not during actual project execution. It is desirable to develop a method of calculating project uncertainty that is computationally simpler, so it could be integrated with project management software and used in all phases of project management.

Applications to Transportation. Transportation infrastructure projects involve a high degree of uncertainty. The usual sources of uncertainty are present in the technical, schedule, and budget aspects of a project, and the uncertainty due to weather, funding issues, and resource availability compound the problem. The current Minnesota funding shortages can cause project delays that affect budgets and schedules. To make things worse, northern transportation projects are done outdoors during a relatively short construction season, so inclement weather and competition for finite contract labor and construction equipment can wreak havoc with schedules and budgets.

Project uncertainty management, if effectively applied to transportation infrastructure projects, could improve decision making and project execution. Better decisions based on a full knowledge of uncertainties and their impact on eventual project outcomes will result in a greater likelihood that the best project and alternatives are chosen. With more detailed information about the sources of uncertainty, risks can be identified, assessed, and reduced. Continuously updating uncertainty information as a project executes will enable better management of the budget, schedule, and resources.

Chapter 2
Project Networks and Uncertainty

Project managers have traditionally relied upon network methods like the Program Evaluation and Review Technique (PERT) and the Critical Path Method (CPM) for planning and managing projects. Many books on project management present the PERT/CPM techniques [e.g. 6, 7, 8, 9, 10], and virtually all project management software packages implement some form of PERT/CPM. These techniques rely on estimates of the expected duration and expected cost of each project task as input, and generate project schedules and budgets based on the logical predecessor/successor relationships between the tasks.

Network structure. Formally, a PERT/CPM project network consists of a set of tasks (nodes) connected by predecessor-successor relationships usually represented as arrows. A project network has two unique nodes, a Start (initial) node that has no predecessors and one or more successors, and an End (terminal) node that has one or more predecessors and no successors. All of the other nodes in a project network are called internal nodes and they represent tasks. Each task must have at least one predecessor (no spontaneous tasks) and one successor (no dead ends). Each internal node is usually assigned a value or weight. In the case of a project network, the node value is the task duration.

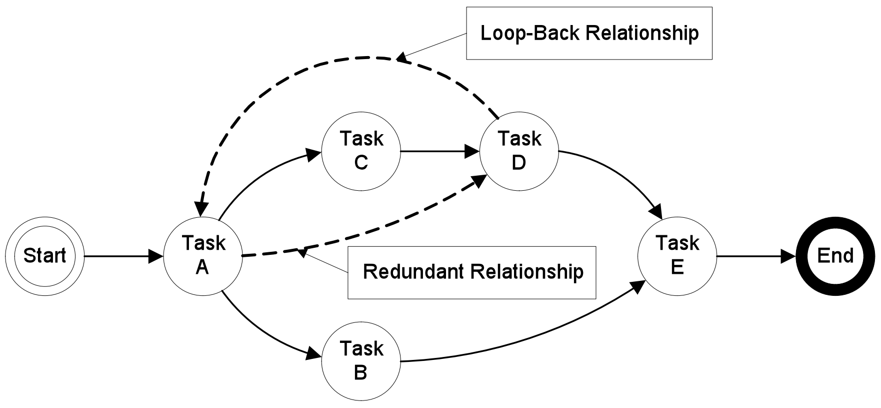

Figure 2.1. Project Network Structure

A path is said to exist between two nodes A and B if node B can be reached form node A by following predecessor-successor relationships. In the example network shown in Figure 2.1, two paths exist between the Start and End nodes (Start-A-C-D-E-End and Start-A-B-E-End). The length of a path is the sum of the durations of each of the tasks encountered as the path is traversed. In a project network, the longest path from the Start to the End node is called the *critical path*.

A legitimate PERT/CPM network is also *acyclic* and *compact* [11]. In an acyclic network, no path exists that passes through any task more than once. This eliminates the possibility of loop-backs. In Figure 2.1, the link from task D to task A is not allowed because it is a loop-back relationship that creates a cycle.

A compact network contains no redundant relationships. This means that only immediate predecessors are shown in a project network. In Figure 2.1 the link from task A to task D is not allowed because it represents a redundant relationship.

Chapter 3
Quantification of Uncertainty

Probability distributions are usually used to represent uncertain task durations. In this research uncertain task durations were represented by discrete random variables with a minimum duration, a maximum duration, and several duration increments between the minimum and maximum values. The number of duration increments is determined by a value defined as the *precision*. The probability of each possible duration occurring is determined by a probability distribution that is assigned to the particular task, and as usual the total probability of all possible outcomes must be 1. The triangle distribution was used to represent task duration uncertainty in this research, but any finite distribution could be used.

It is often desirable to compute the effects of individual task uncertainties on the overall project outcome. For example, given triangle probability distributions that represent uncertain task durations, it is desirable to combine those individual uncertain durations in some way to compute the overall project duration, and the uncertainty therein. Two operators, a series operator and a parallel operator, were developed to help accomplish this.

Two tasks are defined as being in *series* if the first task has the second task as its only successor, and the second task has the first task as its only predecessor. For example, tasks E and F in Figure 3.1 are in series.

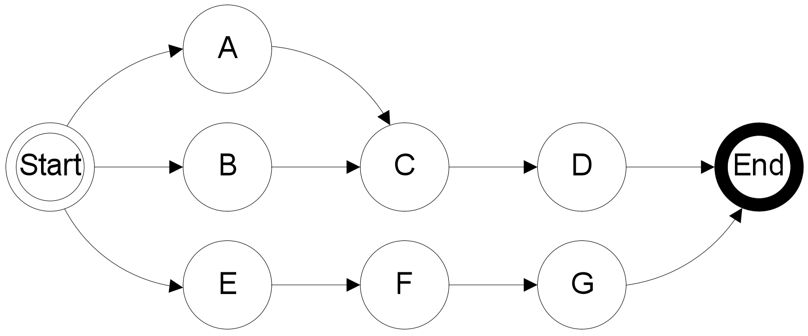

Figure 3.1. Example Project Network

When task durations are certain, the series result can be determined by simply adding the two durations. When durations are represented by discrete random variables, however, all possible outcomes must be determined, along with the probability of each. The possible outcomes are determined by adding all of the possible combinations, while the joint probabilities of each of the combinations are determined by multiplying the marginal probabilities for each task.

A simplified example is shown in Table 3.1. Task E has possible durations of 10, 11, and 12 days with the marginal probabilities shown, and task F has possible durations of 9, 10, and 11 days, each with the probabilities shown. The resulting series outcomes, along with the probability of each, are shown in the table. Note that some outcomes (e.g. 20, 21, and 22) occur more than once, so the total probability of those outcomes must be determined by adding.

Table 3.1. Example Series Computation

E \ F	**9 (.30)**	**10 (.40)**	**11 (.30)**
10 (.20)	19 (.06)	20 (.08)	21 (.06)
11 (.50)	20 (.15)	21 (.20)	22 (.15)
12 (.30)	21 (.09)	22 (.12)	23 (.09)

All possible outcomes, and their probabilities, are shown in Table 3.2. This represents the total duration of the series combination of tasks A and B, and the uncertainty therein.

Table 3.2. Results of Series Computation

Duration	**Probability**
19	.06
20	.23
21	.35
22	.27
23	.09

Two tasks are defined as being in *parallel* if they have identical predecessors and successors. For example, tasks A and B in Figure 3.1 are in parallel.

When task durations are certain, the parallel result can be determined by simply taking the maximum of the two durations. As in the series case, however, all possible outcomes must be determined when durations are represented by discrete random variables, along with the probability of each. The possible outcomes are determined by taking the maximum of all of the possible combinations, while the joint probabilities are determined once again by multiplying the marginal probabilities for each task.

Table 3.3. Example Parallel Computation

A \ B	**9 (.30)**	**10 (.40)**	**11 (.30)**
10 (.20)	10 (.06)	10 (.08)	11 (.06)
11 (.50)	11 (.15)	11 (.20)	11 (.15)
12 (.30)	12 (.09)	12 (.12)	12 (.09)

A simplified example is shown in Table 3.3. Task A has possible durations of 10, 11, and 12 days with the marginal probabilities shown, and task B has possible durations of 9, 10, and 11 days, each with the probabilities shown. The resulting parallel outcomes, along with the probability of each, are shown in the table. As before, the probabilities of outcomes that occur more than once are determined by adding.

All possible outcomes, and their probabilities, are shown in Table 3.4. This represents the total duration of the parallel combination of tasks A and B, and the uncertainty therein.

Table 3.4. Results of Parallel Computation

Duration	**Probability**
10	.14
11	.56
12	.30

A procedure was developed that applies the series and parallel operators repeatedly to a network to compute the overall project duration and uncertainty. The procedure first searches for a series relationship between two tasks, and computes the result if one is found. The network is then searched for a parallel relationship between two tasks, and again the result is computed if such a relationship is found. The procedure is repeated until neither a series nor parallel relationship is found.

The result of applying the procedure to the network in Figure 3.1 is shown in Figure 3.2. Note that tasks are combined in serial and/or parallel until the network is reduced to a single task. The duration of this task is the overall project duration, and the uncertainty in this task represents the overall uncertainty in the project duration.

Computation of slack times. In a project network without uncertainty, slack is defined as the difference in duration between the critical path and a given path. Tasks on the critical path have zero slack, while all others have positive slack. In an uncertain network the computation of slack becomes far more complex. There may be many possible critical paths in an uncertain network, each with some probability of being critical.

In order to calculate the slack for a particular task, its uncertain duration must be compared with all other possible parallel paths through the network. To accomplish this, one more operator, the *negate* operator, was required. This operator essentially reverses the time direction of a task duration and its uncertainty, so uncertain durations can be 'subtracted' in series as well as added.

Consider, for example, the calculation of slack for task B in the project network shown in Figure 3.1. This task can be compared with task A to see the probability that task B will be critical or have slack, but the effect of the uncertainties in the other tasks (C, D, E, F, and G) must also be considered. This is accomplished by first combining tasks E, F, and G in series, and then combining the result in series with the negation of tasks D and C. The resultant task is then effectively in parallel with task A, so they can be combined using the parallel operator. The result can then be compared with task B, again using the negation operator, to determine the slack for task B. This process is illustrated graphically in Figure 3.3.

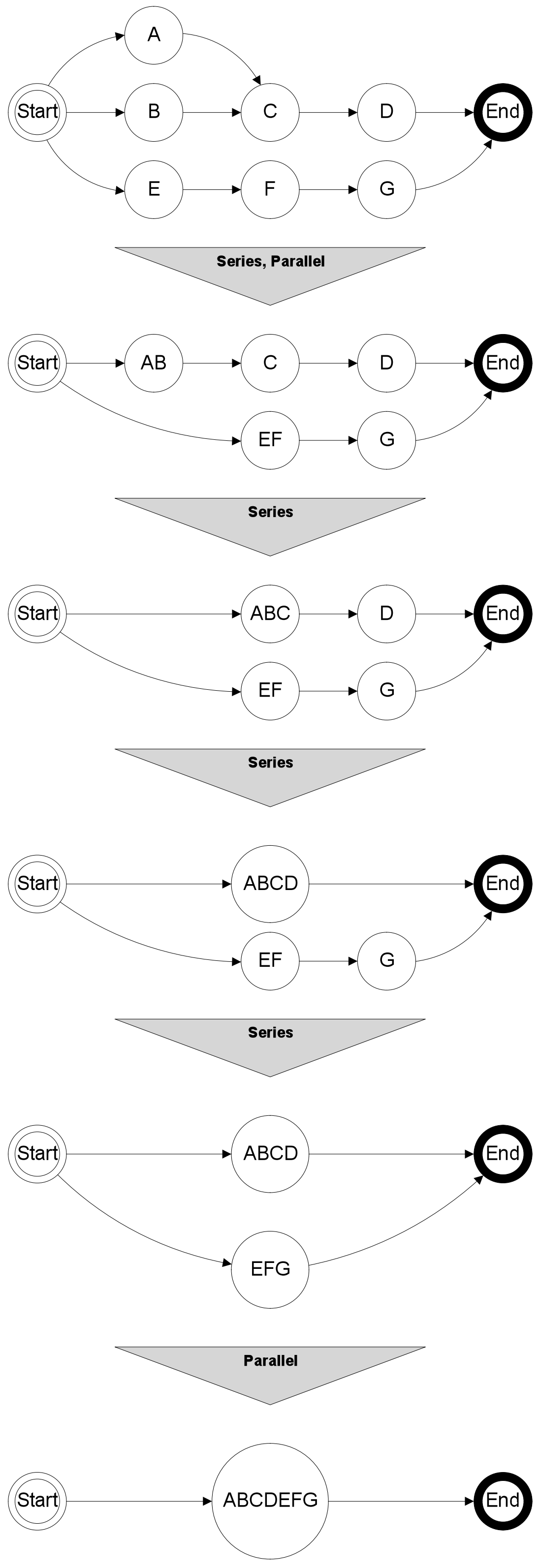

Figure 3.2. Illustration of Network Uncertainty Computation

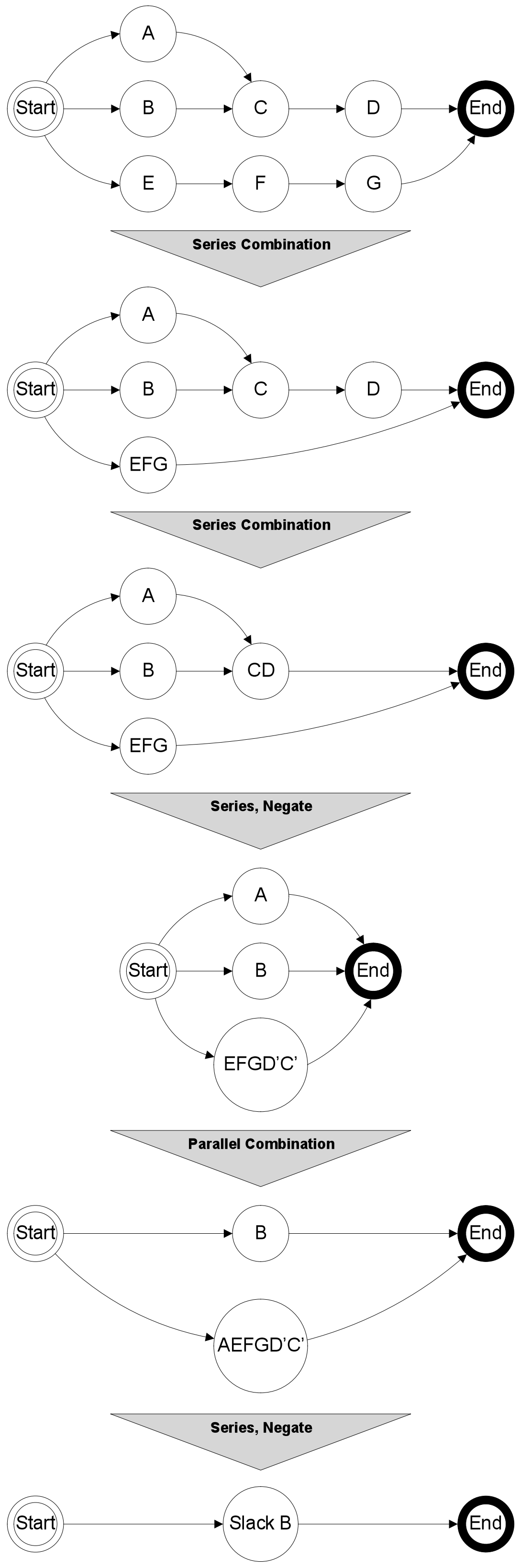

Figure 3.3. Illustration of Slack Computation

Because slack is the difference between two uncertain task durations it can take on both positive and negative values. Tasks with completely negative slack are always critical, and tasks with completely positive slack are never critical. Tasks with slack on both sides of zero have some probability of being critical, and some probability of being non-critical. The area beneath the negative portion of the slack curve is defined as *criticality*, the probability of the task being critical.

Evaluation of the computational methodology. The computational technique described above was tested against Monte Carlo simulation. The method was tested on many networks, and used to calculate many uncertain quantities including task start times, task end times, task slack and criticality, and overall project duration. The results of these computations were compared to the simulation results for the same networks.

Results. The results of applying the computational method to the example network shown in Figure 3.1 are discussed next. These results are representative of the results achieved with the many other network configurations that were tested.

Triangle probability distributions were used to represent the uncertainties in the task durations. The minimum duration, maximum duration, and most likely duration for each task are shown in Table 3.5.

Table 3.5. Minimum, Most Likely (ML) and Maximum Values for Tasks Used in Example

Task	Min.	ML	Max.
A	5	6	8
B	4	5	7
C	4	5	7
D	4	5	7
E	4	5	7
F	4	5	7
G	5	6	8

The overall duration of the project and its uncertainty were calculated using the series and parallel operators as previously shown in Figure 3.2. The calculated results were then compared to the results of three Monte Carlo simulations, with 1000, 10000, and 100000 iterations. The results of this comparison are shown graphically in Figure 3.4.

The mean absolute differences between each of the simulated results and the calculated results were also generated, and they are shown in Table 3.6. Note that the results agree more closely, and the differences become smaller, as the number of iterations in the simulation increases. This would suggest that the calculated result, generated in a single pass, is more accurate then the Monte Carlo simulation with many iterations. It would also suggest that, as the number of iterations becomes very large, the simulation results in fact approach the calculated result as a limit.

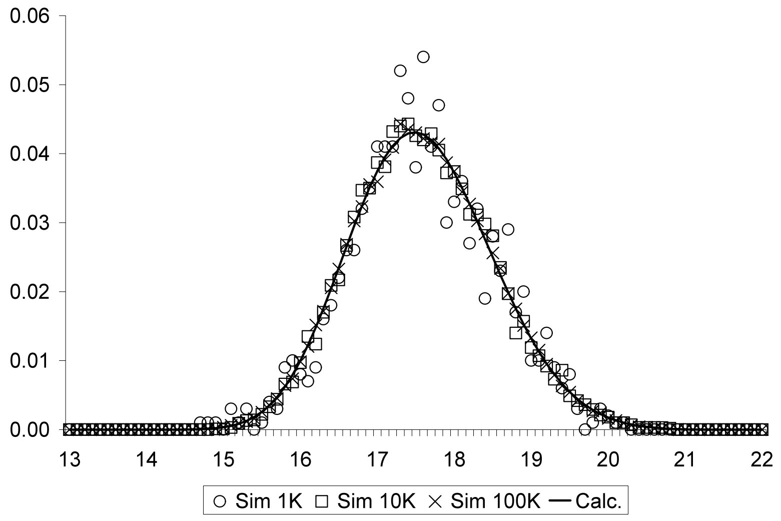

Figure 3.4. Calculated and Simulated Results of Project Duration Uncertainty

Table 3.6. Comparison between Calculated and Simulated Results

Iterations	Mean Absolute Difference
1,000	.001722
10,000	.000515
100,000	.000194

The uncertain slack times, and their related criticalities, were calculated using the series, parallel, and negate operators as previously illustrated in Figure 3.3. The calculated results for task B were then compared to the results of a Monte Carlo simulation with 10000 iterations. The results of this comparison for the task B slack computation and simulation are shown graphically in Figure 3.5.

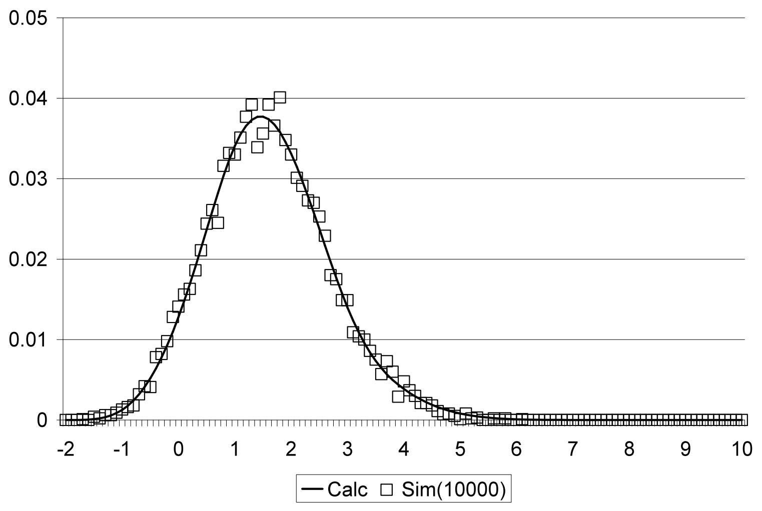

Figure 3.5. Calculated and Simulated Results of Slack for Task B

The computed slack results for each of the tasks were also examined in order to determine the effects of the series and parallel relationships on slack times. The results indicated that, for task in series, slack times and criticalities are identical.

The results also indicated that when tasks are in parallel, criticality divides among the various parallel paths. This fact is demonstrated in Figure 3.6, where the criticality results for all of the tasks in the network are shown. Note also that the total criticality at the Start and End nodes sums to 100%.

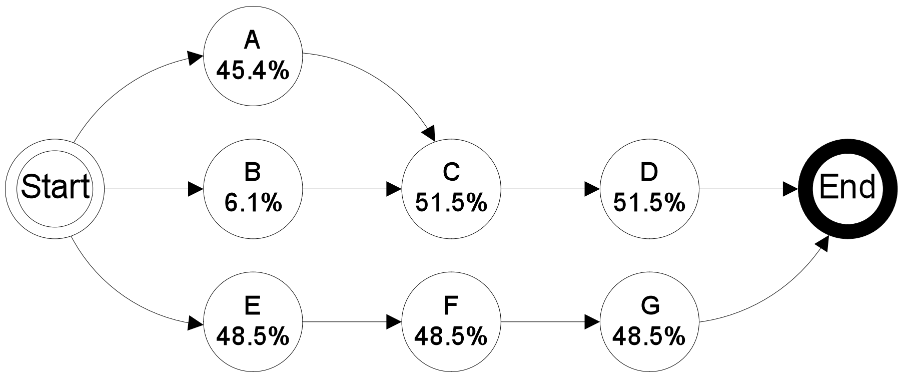

Figure 3.6. Criticality Results for All Network Tasks

Limitations. A major limitation of the computational method described so far is the fact that the method requires that the project network be completely simplified using the series and parallel operators. Although this is possible for many practical networks, there are also many networks for which simplification is not possible due to path dependencies. An enhancement to the computational method that deals with path dependencies is discussed in the following chapter.

Chapter 4
Computation of Uncertainty with Dependencies

A seemingly simple change to a project network can introduce dependencies that make the computation of uncertainty difficult. Figure 4.1 shows the example project network that has was used in Chapter 3 with one additional relationship added from task C to task G. Note that the network can only be partially simplified with the series and parallel operators, because task G participates in two paths. Task G could be replicated (one copy in parallel with D and one in series with EF) and the resulting network could apparently be simplified, but when the final parallel combination was performed it would be between two dependent paths (both contain the uncertainty in task G), and the series and parallel operators assume independence. At this point a different technique must be used to determine the overall uncertainty in the partially simplified network.

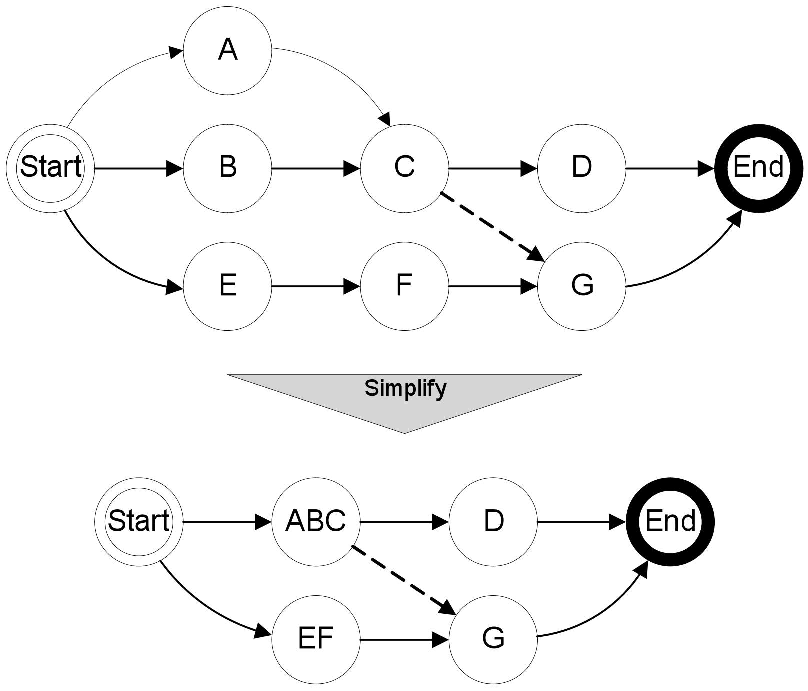

Figure 4.1. Project Network with Path Dependencies

Replicating a Task. As previously mentioned, the first step in solving a network that has path dependencies is to replicate tasks that participate in two or more paths. Once the original network has been simplified as far as possible using the series and parallel operators, the simplified network is searched to find tasks that need to be replicated. The method used is to simply find any task that has multiple predecessors, and create a replica for each predecessor. This search-and-replicate procedure is repeated until no tasks with multiple predecessors are found. This guarantees that the resulting network, with the replica tasks inserted, can be completely simplified using the series and parallel operators. The result of applying this procedure to the example network is shown in Figure 4.2. The resulting network can clearly be simplified using the series and parallel operators, if the dependences between tasks G1 and G2 can be avoided.

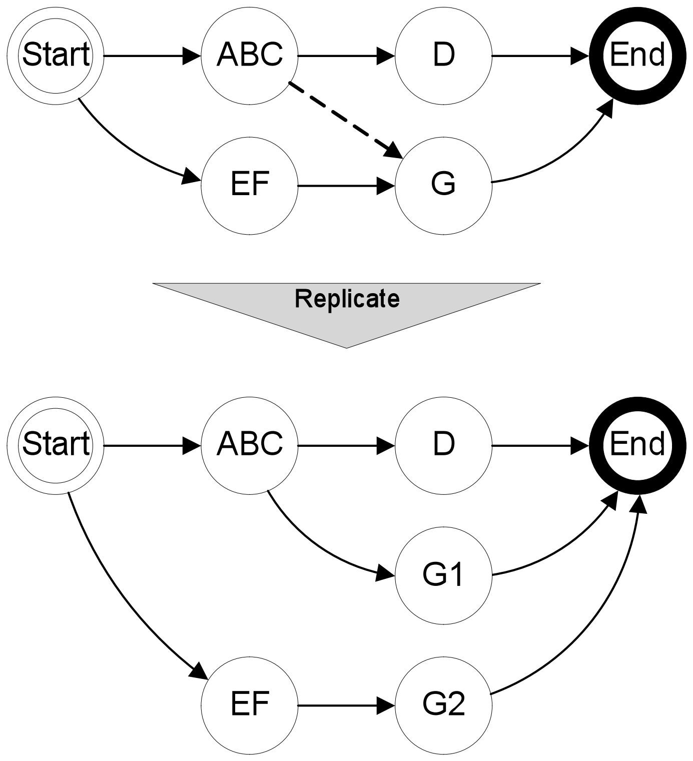

Figure 4.2. Result of Task Replication

When a task is replicated, the original task is placed in a list of replicated tasks, complete with its uncertain duration. One replica of the original task is then generated for each predecessor, and inserted into the network. The replica tasks each have one predecessor, and have the same successors as the original task. Their names are generated by concatenating a serial number onto the original task name. The replicas differ from the original in one other respect – they have no uncertainty. They are allowed to take on only one certain (fixed) duration at any given time. This avoids the problem of dependence between parallel paths.

In order to simplify a network that contains replica tasks, the series and parallel operators were modified slightly so they could accurately deal with any combination of certain and uncertain durations. The uncertainty in the project duration was then computed by compiling the results of the simplifications for each possible value of the replicated tasks, and combing them considering the probability of each occurrence.

In the example above, replica tasks G1 and G2 were inserted into the network. They were then each be assigned the same certain duration value, drawn from the possible duration values for the original task G. The network was then simplified and the resulting uncertain duration generated. The results were then modified considering the probability of that particular duration of task G occurring. This procedure was repeated for every possible duration value for task G, from the minimum to the maximum with the task precision determining the step size. All of the results were then combined to determine the overall project duration uncertainty. The result of this computation is shown in Figure 4.3, and compared to Monte Carlo simulation results for the original network after 50,000 iterations.

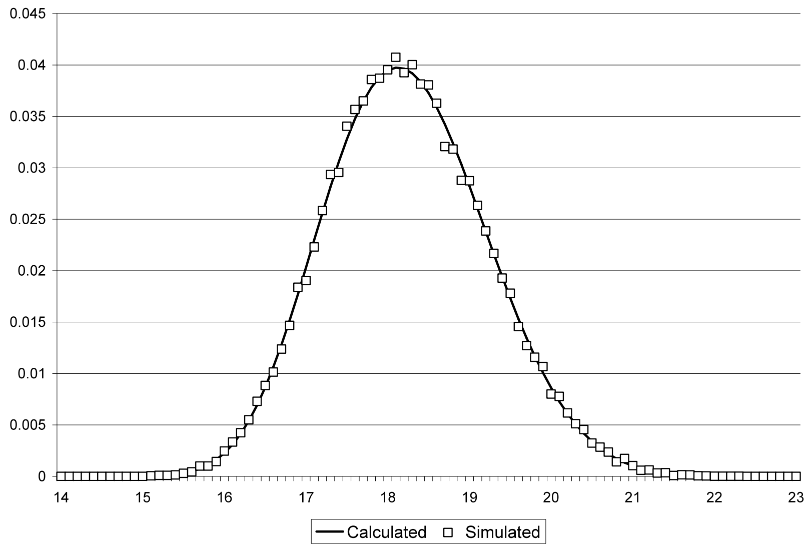

Figure 4.3. Calculated and Simulated Results for Project Duration

Replicating Multiple Tasks. A more complex example is now presented to illustrate the situation in which more than one task needs to be replicated. The network that will be analyzed is shown in Figure 4.4. This has a very complex structure with multiple dependencies, and it has been extensively studied in the literature [e.g. 11, 12, 13]. Note that the original network cannot be simplified at all using the series and parallel operators.

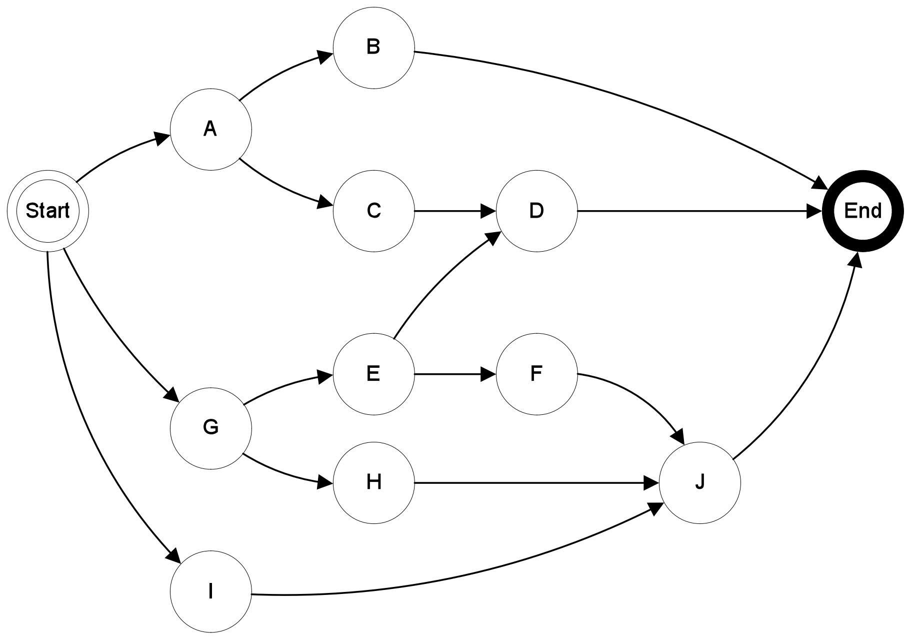

Figure 4.4. Complex Example Network

As before, triangle probability distributions were used to represent the uncertainties in the task durations in this network. The minimum duration, maximum duration, and most likely duration for each task are shown in Table 4.1.

Table 4.1. Minimum, Most Likely (ML) and Maximum Values for Tasks Used in Example

Task	Min.	ML	Max.
A	9	10	12
B	29	30	32
C	9	10	12
D	18	19	21
E	9	10	12
F	9	10	12
G	9	10	12
H	19	20	22
I	29	30	32
J	8	9	11

Although this network appears to contain many dependencies, only two tasks, task D and task J, have multiple predecessors and need to be replicated. When the search-and-replicate procedure described previously is applied, tasks D and J are added to the list of replicated tasks, while their replicas are inserted into the network as necessary. The network with the replica tasks inserted is shown in Figure 4.5

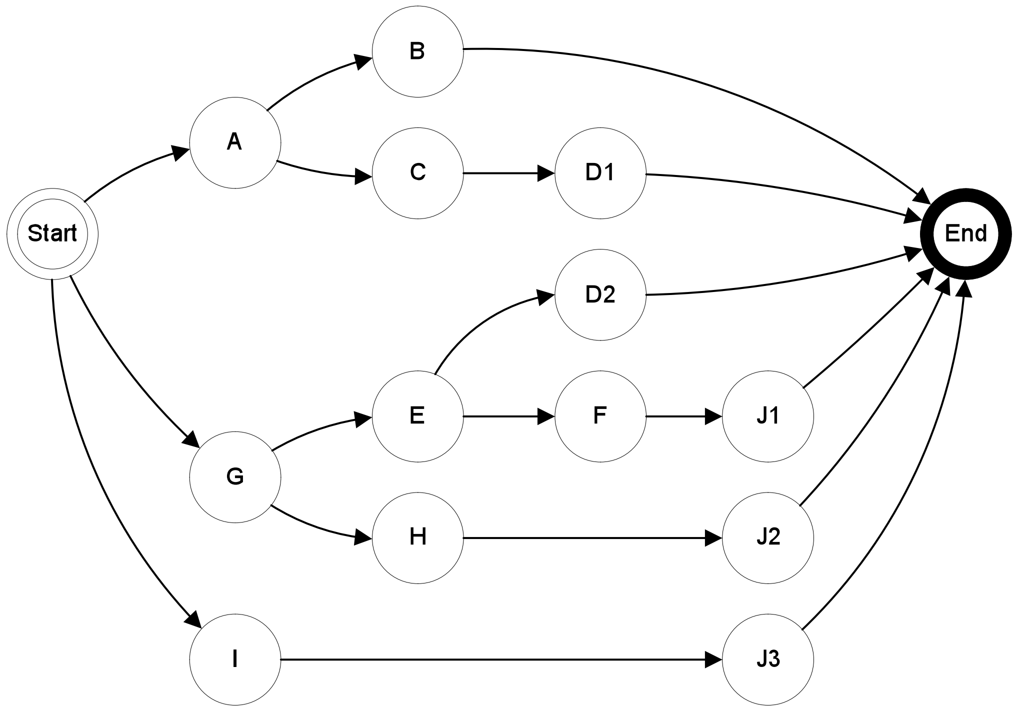

Figure 4.5. Complex Example Network with Replica Tasks Inserted

As can be seen in Figure 4.5, the network can be fully simplified by repeatedly applying the series and parallel operators. As before, however, the replica tasks must be assigned fixed values drawn from the duration distributions of the original tasks to avoid dependencies. To accomplish this, all possible combinations of the durations of tasks D and J must be generated, and the resulting network simplified each time. As before, the results of the many simplifications must be combined, considering the joint probability of each combination occurring, to compute the final overall project duration uncertainty. The result of this computation is shown in Figure 4.6, and compared to Monte Carlo simulation results for the original network after 50,000 iterations.

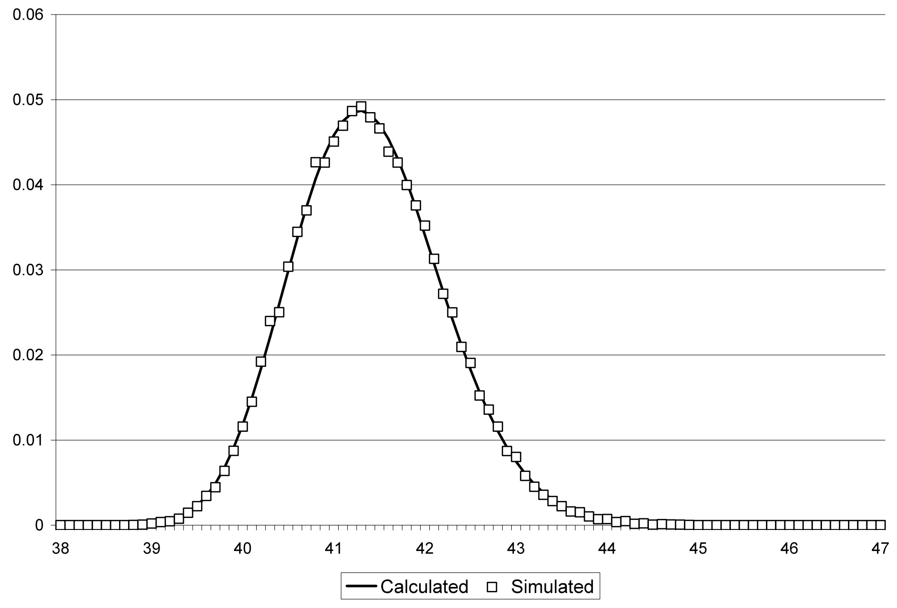

Figure 4.6. Calculated and Simulated Results for Complex Project Duration

Limitations. The only practical limitation on the method presented here is the fact that once the replication of tasks becomes necessary, the computation time required becomes exponential in the number of replicated tasks. For the examples presented here, the computation time with no replicated tasks (Figure 3.4) is about a second, with one replicated task (Figure 4.3) a few seconds, and with two replicated tasks (Figure 4.6) somewhat over a minute. Further investigation demonstrated that networks requiring three tasks to be replicated were computed in about 45 minutes, and if four tasks were to be replicated the computation would take on the order of a day, and five replicated tasks would take on the order of a month to compute.

Very good results can be achieved using Monte Carlo simulation with about 200,000 iterations, and this requires a few minutes of computing time on the example networks presented here. It is likely that a hybrid technique that uses the serial and parallel operators to simplify the project network as much as possible, and then employs Monte Carlo simulation on the simplified network would be the most efficient approach to computing overall project duration uncertainty.

Chapter 5
Conclusions

The conclusions based on the evaluation of the computational methodology presented in this report can be summarized as follows:

- The method accurately computes the effect of uncertainties in individual task durations on the overall project duration in project networks.
- The method accurately computes the uncertainty distributions in task slack times, and the resulting task criticality.
- Task criticality is in fact the area beneath the negative portion of the slack time distribution.
- Slack time distributions for tasks in series are identical.
- Criticality divides among parallel paths
- The total criticality for the project, as computed at the Start and End nodes, is 100%.
- The method requires exponential computational time on project networks with path dependencies, but can be used to partially simplify those networks to facilitate more rapid Monte Carlo simulation.

References

1. Raz, T. and E. Michael, "Use and Benefits of Tools for Project Risk Management," *International Journal of Project Management*, Vol. 19 (2001), 9-17.

2. PMI, *A Guide to the Project Management Body of Knowledge* (Project Management Institute, 2000).

3. Risk Specific Interest Group (RiskSIG) of the Project Management Institute (Internet), *Risk Management Tools* (cited Feb. 2003), http://www.risksig.com/resource/index.htm.

4. Williams, T. M., "A Classified Bibliography of Recent Research Relating to Project Risk Management," *European Journal of Operational Research*, Vol. 85 (1995), 18-38.

5. Ward, S. and C. Chapman, "Transforming Project Risk Management into Project Uncertainty Management," *International Journal of Project Management*, Vol. 21 (2003), 97-105.

6. Cleland, D. I. and W. R. King (eds.), *Project Management Handbook*, second edition (Van Nostrand Reinhold, 1988).

7. Kerzner, H. (ed.), *Project Management - A Systems Approach to Planning, Scheduling, and Controlling*, seventh edition (John Wiley and Sons, 2001).

8. Meridith, Jack R. and Samuel J. Mantel Jr., *Project Management - A Managerial Approach*, fourth edition (John Wiley and Sons, 2000).

9. Rosenau, Milton D., *Successful Project Management - A Step-by-Step Approach with Practical Examples*, second edition (Van Nostrand Reinhold, 1992).

10. Stuckenbruck, Linn C. (ed.), *The Implementation of Project Management: The Professional's Handbook* (Addison-Wesley, 1981).

11. Nasution, S. H., "Fuzzy Critical Path Method," *IEEE Transactions on Systems, Man, and Cybernetics,* Vol. 24 (1994), 48-57.

12. Dubois, D. J. and Prade, H., *Fuzzy Sets and Systems: Theory and Applications* (New York: Academic Press, 1980).

13. Dubois, D. J. and Prade, H., *Possibility Theory: An Approach to Computerized Processing of Uncertainty* (New York: Plenum, 1988).

Appendix A

Computer Program Listing
(Python Language)

Program Project1

```
import random
from random import random
# Classes
class U_time:
    def __init__(self):
        self.min=0.0
        self.max=0.0
        self.prec=0.1
        self.pdf=[]

    def print_me(self):
        total=0.0
        for val in self.pdf:
            total=total+val
            print '%6.4f'%(val)
        print 'Total:',total

    def write_me(self,file_name):
        file_name=file_name+'.csv'
        outfile=open(file_name,'w')
        dur=self.min
        for val in self.pdf:
            tempstr='%8.4f'%dur
            tempstr=tempstr+','+'%12.8f'%val+'\n'
            outfile.write(tempstr)
            dur=dur+self.prec
        outfile.close()
        print 'File ',file_name,' written and closed'

    def triangle(self,a,b,c,p):
        self.min=a
        self.max=c
        self.prec=p
        x=a
        val=0.0
        inc=(2.0*p)/((b-a)*(c-a))
        while x<(b-0.5*p):
            self.pdf.append(val*p)
            val=val+inc
            x=x+p
        val=2.0/(c-a)
        self.pdf.append(val*p)
        inc=(-2.0*p)/((c-b)*(c-a))
        val=val+inc
        x=b+p
        while x<(c-0.5*p):
            self.pdf.append(val*p)
            val=val+inc
            x=x+p
        val=0.0
```

```
            self.pdf.append(val*p)

    def simulate(self):
        pct=float(int(random()*10000.0+0.5))/10000.0
        cum=0.0
        n=0
        while cum<pct:
            cum=cum+self.pdf[n]
            n=n+1
        return self.min+self.prec*(n-1)

    def rtriangle(self,a,c,m,p):
        self.min=a
        self.max=c
        self.prec=p
        x=a
        val=(2.0*m)/(c-a)
        self.pdf.append(val*p)
        inc=(-2.0*p*m)/((c-a)*(c-a))
        val=val+inc
        x=a+p
        while x<(c-0.5*p):
            self.pdf.append(val*p)
            val=val+inc
            x=x+p
        val=0.0
        self.pdf.append(val*p)

    def dummy(self,a,x,p):
        self.min=a
        self.max=a
        self.prec=p
        self.pdf.append(x)

class Project:
    def __init__(self,nm,p):
        self.proj_name=nm
        self.prec=p
        self.net=Network(self.prec)

    def print_me(self,n):
        print '\nProject: ',self.proj_name
        if n==1:
            for taskX in self.net.tasks:
                print '\nTask ',taskX.task_name
                print 'Predecessors:'
                for taskY in taskX.pred:
                    print taskY.task_name
                print 'Successors:'
                for taskY in taskX.succ:
                    print taskY.task_name
        else:
```

```
        for taskX in self.netsimp.tasks:
            print '\nTask ',taskX.task_name
            print 'Predecessors:'
            for taskY in taskX.pred:
                print taskY.task_name
            print 'Successors:'
            for taskY in taskX.succ:
                print taskY.task_name

    def simulate(self,iter):
        self.sdur=self.net.sim_dur(iter)

    def simplify(self,netin,dur):
        #dur should be an empty U_time
        netcpy=Network(self.prec)
        #copy the network
        #first insert copies of tasks
        n=2
        while n<len(netin.tasks):
            taskX=netin.tasks[n]
            taskY=Task(taskX.task_name,0,0,0,taskX.duration.prec)
            taskY.duration.min=taskX.duration.min
            taskY.duration.max=taskX.duration.max
            for val in taskX.duration.pdf:
                 taskY.duration.pdf.append(val)
            taskY.id=len(netcpy.tasks)
            netcpy.tasks.append(taskY)
            n=n+1
        #then create links
        n=0
        while n<len(netin.tasks):
            for succ_t in netin.tasks[n].succ:
                netcpy.add_link(n,netin.tasks.index(succ_t))
            n=n+1
        #simplify the copied network
        found=1
        while found==1:
            found=0
            n=0
            while (found==0)&(n<len(netcpy.tasks)):
                taskX=netcpy.tasks[n]
                #check for serial relationships
                for taskY in taskX.succ:
                    if inSeries(taskX,taskY)>0:
                        found=1
                        netcpy.comb_ser(taskX.id,taskY.id)
                if found==0:
                    #check for parallel relationships
                    i=taskX.id+1
                    while (found==0)&(i<len(netcpy.tasks)):
                        taskY=netcpy.tasks[i]
                        if inParallel(taskX,taskY)>0:
                            found=1
                            netcpy.comb_par(taskX.id,taskY.id)
                        i=i+1
                n=n+1
                #end scan
            #end rescans
        taskno=len(netcpy.tasks)-2
        if taskno==1:
            #duration is known
            dur.max=netcpy.tasks[2].duration.max
            dur.min=netcpy.tasks[2].duration.min
            #empty the pdf
            while len(dur.pdf)>0: del dur.pdf[0]
            for val in netcpy.tasks[2].duration.pdf:
                dur.pdf.append(val)
        else:
            #create a simplified network out of thin air
            self.netsimp=Network(self.prec)
            #copy the network
            #first insert copies of tasks
            n=2
            while n<len(netcpy.tasks):
                taskX=netcpy.tasks[n]
                taskY=Task(taskX.task_name,0,0,0,taskX.duration.prec)
                taskY.duration.min=taskX.duration.min
                taskY.duration.max=taskX.duration.max
                for val in taskX.duration.pdf:
                      taskY.duration.pdf.append(val)
                taskY.id=len(self.netsimp.tasks)
                self.netsimp.tasks.append(taskY)
                n=n+1
            #then create links
            n=0
            while n<len(netcpy.tasks):
                for succ_t in netcpy.tasks[n].succ:
                  self.netsimp.add_link(n,netcpy.tasks.index(succ_t))
                n=n+1
        del netcpy
        return taskno

    def calc_dur(self):
        dur=U_time()
        dur.prec=self.prec
        taskno=self.simplify(self.net,dur)
        if taskno==1:
            #network fully simplified
            self.cdur=dur
        else:
            del dur
            #task duplication required
            #initialize the duration and pdf
            self.cdur=U_time()
```

```
self.cdur.prec=self.prec
self.cdur.max=self.netsimp.tasks[1].get_max()
self.cdur.min=self.netsimp.tasks[1].get_min()
#build empty pdf
nmax=(int)((self.cdur.max-self.cdur.min+
     self.cdur.prec*0.5)/self.cdur.prec)
n=0
while n<=nmax:
    self.cdur.pdf.append(0.0)
    n=n+1
#build list of tasks for joint solver
duplist=[]
found=1
while found==1:
    found=0
    n=2
    while (n<len(self.netsimp.tasks))&(found==0):
        if len(self.netsimp.tasks[n].pred)>1:
            #duplicate this task
            found=1
            taskA=self.netsimp.tasks[n]
            #check if task is a duplicate
            if len(taskA.duration.pdf)<2:
                #duplicating a duplicate
                print 'Duplicating duplicate task',
                     taskA.task_name
                #find original in duplist
                for task1 in duplist:
                    for task2 in task1.dup:
                        if task2.task_name==
                          taskA.task_name: taskD=task1
                m=1
                #duplicate the duplicate as necessary
                for pred_t in taskA.pred:
                    nm=taskA.task_name+str(m)
                    taskC=Task(nm,0,0,0)
                    #fix current predecessor
                    pred_t.succ.remove(taskA)
                    pred_t.succ.append(taskC)
                    taskC.pred.append(pred_t)
                    #correct successors
                    for succ_t in taskA.succ:
                        succ_t.pred.append(taskC)
                        taskC.succ.append(succ_t)
                    # add combined task to project
                    self.netsimp.tasks.append(taskC)
                    taskD.dup.append(taskC)
                    m=m+1
                #remove duplicate successors
                for succ_t in taskA.succ:
                     succ_t.pred.remove(taskA)
                # remove duplicated task
                self.netsimp.tasks.remove(taskA)
                # renumber task ids
                i=0
                while i<len(self.netsimp.tasks):
                    self.netsimp.tasks[i].id=i
                    i=i+1
                    #end renumbering
                #end if duplicate
            else:
                #not a duplicate
                print 'Duplicating task ',taskA.task_name
                duplist.append(taskA)
                m=1
                for pred_t in taskA.pred:
                    nm=taskA.task_name+str(m)
                    taskC=Task(nm,0,0,0,self.prec)
                    taskC.duration.pdf.append(0.0)
                    #fix current predecessor
                    pred_t.succ.remove(taskA)
                    pred_t.succ.append(taskC)
                    taskC.pred.append(pred_t)
                    #correct successors
                    for succ_t in taskA.succ:
                        succ_t.pred.append(taskC)
                        taskC.succ.append(succ_t)
                    # add combined task to project
                    self.netsimp.tasks.append(taskC)
                    taskA.dup.append(taskC)
                    m=m+1
                #remove duplicate successors
                for succ_t in taskA.succ:
                     succ_t.pred.remove(taskA)
                # remove duplicated task
                self.netsimp.tasks.remove(taskA)
                # renumber task ids
                i=0
                while i<len(self.netsimp.tasks):
                    self.netsimp.tasks[i].id=i
                    i=i+1
                    #end renumbering
                #end if not duplicate
            #end if multiple preds
        n=n+1
        #end scan
    #end rescans
#print the duplicates list
print '\nDuplicate task list:'
for taskX in duplist:
    print 'Duplicated task: ',taskX.task_name
    for taskY in taskX.dup:
        print '   Duplicate: ',taskY.task_name
```

```
#joint prob solver
if len(duplist)>3: print 'Too many duplicate tasks.'
elif len(duplist)>2:
    print 'Three duplicated tasks'
    #do three loops
    #outer loop
    dur0=duplist[0].duration.min
    for val0 in duplist[0].duration.pdf:
        #assign duplicates their values
        for dup0 in duplist[0].dup:
            dup0.duration.min=dur0
            dup0.duration.max=dur0
            dup0.duration.pdf[0]=1.0
        dur0=dur0+duplist[0].duration.prec
        #next loop
        dur1=duplist[1].duration.min
        for val1 in duplist[1].duration.pdf:
            #assign duplicates their values
            for dup1 in duplist[1].dup:
                dup1.duration.min=dur1
                dup1.duration.max=dur1
                dup1.duration.pdf[0]=1.0
            dur1=dur1+duplist[1].duration.prec
            #inner loop
            dur2=duplist[2].duration.min
            for val2 in duplist[2].duration.pdf:
                #assign duplicates their values
                for dup2 in duplist[2].dup:
                    dup2.duration.min=dur2
                    dup2.duration.max=dur2
                    dup2.duration.pdf[0]=1.0
                dur2=dur2+duplist[2].duration.prec
                #add this iteration to total
                durX=U_time()
                taskno=self.simplify(self.netsimp,durX)
                if taskno==1:
                    ofs=int((durX.min-self.cdur.min+
                          self.prec*0.5)/self.prec)
                    for val in durX.pdf:
                          self.cdur.pdf[ofs]=
                          self.cdur.pdf[ofs]+
                          val*val0*val1*val2)
                        ofs=ofs+1
                else:
                    print 'Network not simplified,
                          taskno is ',taskno
                del durX
elif len(duplist)>1:
    print 'Two duplicated tasks'
    #do two loops
    #outer loop
    dur0=duplist[0].duration.min
    for val0 in duplist[0].duration.pdf:
        #assign duplicates their values
        for dup0 in duplist[0].dup:
            dup0.duration.min=dur0
            dup0.duration.max=dur0
            dup0.duration.pdf[0]=1.0
        dur0=dur0+duplist[0].duration.prec
        #inner loop
        dur1=duplist[1].duration.min
        for val1 in duplist[1].duration.pdf:
            #assign duplicates their values
            for dup1 in duplist[1].dup:
                dup1.duration.min=dur1
                dup1.duration.max=dur1
                dup1.duration.pdf[0]=1.0
            dur1=dur1+duplist[1].duration.prec
            #add this iteration to the total
            durX=U_time()
            taskno=self.simplify(self.netsimp,durX)
            if taskno==1:
                ofs=int((durX.min-self.cdur.min+
                     self.prec*0.5)/self.prec)
                for val in durX.pdf:
                    self.cdur.pdf[ofs]=
                      self.cdur.pdf[ofs]+(val*val0*val1)
                    ofs=ofs+1
            else:
                print 'Network not simplified,
                          taskno is ',taskno
            del durX
else:
    print 'One duplicated task'
    #do one loop
    #outer loop
    dur0=duplist[0].duration.min
    for val0 in duplist[0].duration.pdf:
        #assign duplicates their values
        for dup0 in duplist[0].dup:
            dup0.duration.min=dur0
            dup0.duration.max=dur0
            dup0.duration.pdf[0]=1.0
        dur0=dur0+duplist[0].duration.prec
        #add this iteration to the total
        durX=U_time()
        taskno=self.simplify(self.netsimp,durX)
        if taskno==1:
            ofs=int((durX.min-self.cdur.min+
                self.prec*0.5)/self.prec)
            for val in durX.pdf:
                self.cdur.pdf[ofs]=
                     self.cdur.pdf[ofs]+(val*val0)
                ofs=ofs+1
```

```
                    else: print 'Network not simplified,
                                 taskno is ',taskno
                    del durX

class Network:
    def __init__(self,p):
        self.tasks=[]
        self.prec=p
        self.add_task('START',0,0,0)
        self.add_task('END',0,0,0)

    def add_task(self,nm,a,b,c):
        taskA=Task(nm,a,b,c,self.prec)
        taskA.id=len(self.tasks)
        self.tasks.append(taskA)

    def add_link(self,pred_id,succ_id):
        self.tasks[pred_id].add_succ(self.tasks[succ_id])
        self.tasks[succ_id].add_pred(self.tasks[pred_id])

    def comb_ser(self,idA,idB):
        taskA=self.tasks[idA]
        taskB=self.tasks[idB]
        nmC=self.tasks[idA].task_name+self.tasks[idB].task_name
        taskC=Task(nmC,0,0,0,self.prec)
        taskC.duration=series(taskA.duration,taskB.duration)
        # fix predecessors
        for pred_t in taskA.pred:
            pred_t.succ.remove(taskA)
            pred_t.succ.append(taskC)
            taskC.pred.append(pred_t)
        # fix successors
        for succ_t in taskB.succ:
            succ_t.pred.remove(taskB)
            succ_t.pred.append(taskC)
            taskC.succ.append(succ_t)
        # add combined task to project
        self.tasks.append(taskC)
        # remove old tasks
        self.tasks.remove(taskA)
        self.tasks.remove(taskB)
        # renumber task ids
        n=0
        while n<len(self.tasks):
            self.tasks[n].id=n
            n=n+1

    def comb_par(self,idA,idB):
        taskA=self.tasks[idA]
        taskB=self.tasks[idB]
        nmC=self.tasks[idA].task_name+self.tasks[idB].task_name
        taskC=Task(nmC,0,0,0,self.prec)
        taskC.duration=parallel(taskA.duration,taskB.duration)
        # fix predecessors
        for pred_t in taskA.pred:
            pred_t.succ.remove(taskA)
            pred_t.succ.remove(taskB)
            pred_t.succ.append(taskC)
            taskC.pred.append(pred_t)
        # fix successors
        for succ_t in taskA.succ:
            succ_t.pred.remove(taskA)
            succ_t.pred.remove(taskB)
            succ_t.pred.append(taskC)
            taskC.succ.append(succ_t)
        # add combined task to project
        self.tasks.append(taskC)
        # remove old tasks
        self.tasks.remove(taskA)
        self.tasks.remove(taskB)
        # renumber task ids
        n=0
        while n<len(self.tasks):
            self.tasks[n].id=n
            n=n+1

    def sim_dur(self,iter):
        sdur=U_time()
        #calculate max and min
        sdur.max=self.tasks[1].get_max()
        sdur.min=self.tasks[1].get_min()
        sdur.prec=self.prec
        #build empty pdf
        nmax=(int)((sdur.max-sdur.min+sdur.prec*0.5)/sdur.prec)
        n=0
        while n<=nmax:
            sdur.pdf.append(0.0)
            n=n+1
        #perform the simulations
        i=0
        while i<iter:
            for t in self.tasks: t.simval()
            val=self.tasks[1].resolve()
            ind=int((val-sdur.min+0.05)/sdur.prec)
            sdur.pdf[ind]=sdur.pdf[ind]+1.0/float(iter)
            i=i+1
        return sdur

class Task:
    def __init__(self,nm,a,b,c,p):
        self.task_name=nm
        self.duration=U_time()
        self.duration.prec=p
        if a != 0 or b != 0 or c != 0:
           self.duration.triangle(a,b,c,self.duration.prec)
        self.pred=[]
        self.succ=[]
        self.dup=[]
        self.id=0
        self.simdur=0.0
    def add_pred(self,t):
        self.pred.append(t)
    def add_succ(self,t):
        self.succ.append(t)
    def get_max(self):
        mymax=self.duration.max
        if self.pred[0].id==0: return mymax
        else:
            predmax=0.0
            for taskX in self.pred:
                maxval=taskX.get_max()
                if maxval>predmax: predmax=maxval
            return mymax+predmax
    def get_min(self):
        mymin=self.duration.min
        if self.pred[0].id==0: return mymin
        else:
            predmin=0.0
            for taskX in self.pred:
                minval=taskX.get_min()
                if minval>predmin: predmin=minval
            return mymin+predmin

    def simval(self):
        if self.id>1: self.simdur=self.duration.simulate()

    def resolve(self):
        if self.pred[0].id==0:
            return self.simdur
        elif len(self.pred)==1:
            return self.simdur+self.pred[0].resolve()
        else:
            p1=max(self.pred[0].resolve(),self.pred[1].resolve())
            n=2
            while n<len(self.pred):
                p1=max(p1,self.pred[n].resolve())
                n=n+1
            if self.id==1: return p1
            else: return self.simdur+p1


# Global funtions
def max(a,b):
    if a>b:
        return a
    else:
        return b

def series(ta,tb):
    tc=U_time()
    tc.min=ta.min+tb.min
    tc.max=ta.max+tb.max
    tc.prec=ta.prec
    if len(ta.pdf)==1:
        for val in tb.pdf:
            tc.pdf.append(val)
    elif len(tb.pdf)==1:
        for val in ta.pdf:
            tc.pdf.append(val)
    else:
        nmax=(int)((tc.max-tc.min+tc.prec*0.5)/tc.prec)
        n=0
        while n<=nmax:
            tc.pdf.append(0.0)
            n=n+1
        dura=ta.min
        for vala in ta.pdf:
            durb=tb.min
            for valb in tb.pdf:
                durc=dura+durb
                indc=(int)((durc-tc.min+tc.prec*0.5)/tc.prec)
                tc.pdf[indc]=tc.pdf[indc]+vala*valb
                durb=durb+tb.prec
            dura=dura+ta.prec
    return tc

def parallel(ta,tb):
    tc=U_time()
    if ta.min>=tb.max:
        tc.min=ta.min
        tc.max=ta.max
        tc.prec=ta.prec
        for val in ta.pdf:
            tc.pdf.append(val)
    elif tb.min>=ta.max:
        tc.min=tb.min
        tc.max=tb.max
        tc.prec=tb.prec
        for val in tb.pdf:
            tc.pdf.append(val)
    else:
        tc.min=max(ta.min,tb.min)
        tc.max=max(ta.max,tb.max)
        tc.prec=ta.prec
        nmax=(int)((tc.max-tc.min+tc.prec*0.5)/tc.prec)
        n=0
        while n<=nmax:
            tc.pdf.append(0.0)
            n=n+1
        dura=ta.min
        for vala in ta.pdf:
            durb=tb.min
            for valb in tb.pdf:
                durc=max(dura,durb)
                indc=(int)((durc-tc.min+tc.prec*0.5)/tc.prec)
                tc.pdf[indc]=tc.pdf[indc]+(vala*valb)
                durb=durb+tb.prec
            dura=dura+ta.prec
    return tc

def negate(ta):
    tn=U_time()
    tn.max=-1.0*ta.min
    tn.min=-1.0*ta.max
    tn.prec=ta.prec
    for val in ta.pdf:
        tn.pdf.insert(0,val)
    return tn

def inSeries(taskA,taskB):
    found=0
    if (taskA.id!=0)&(taskB.id!=1):
        if len(taskA.succ)==1:
            if len(taskA.succ[0].pred)==1:
                found=1
    return found

def inParallel(taskA,taskB):
    found=0
    if (len(taskA.pred)==len(taskB.pred))&(len(taskA.succ)==
      len(taskB.succ)):
        # check the predecessors
        mismatch=0
        for taskX in taskA.pred:
            if taskB.pred.count(taskX)<1: mismatch=mismatch+1
        if mismatch==0:
            # check the successors
            for taskX in taskA.succ:
                if taskB.succ.count(taskX)<1: mismatch=mismatch+1
        if mismatch==0:
            found=1
    return found

# Main function
def main():
    proj=1
    if proj>0:
        # create the project task list - START and END are id0 and id1
        test_prj=Project('Test1',0.1)
        test_prj.net.add_task('A',5,6,8)  #id2
        test_prj.net.add_task('B',4,5,7)  #id3
        test_prj.net.add_task('C',4,5,7)  #id4
        test_prj.net.add_task('D',4,5,7)  #id5
        test_prj.net.add_task('E',4,5,7)  #id6
        test_prj.net.add_task('F',4,5,7)  #id7
        test_prj.net.add_task('G',5,6,8)  #id8
        #test_prj.net.add_task('H',19,20,22) #id9
        #test_prj.net.add_task('I',29,30,32) #id10
        #test_prj.net.add_task('J',8,9,11)   #id11
        #test_prj.net.add_task('K',8,9,11)   #id12
        # add links
        test_prj.net.add_link(0,2)
        test_prj.net.add_link(0,3)
        test_prj.net.add_link(0,6)
        test_prj.net.add_link(2,4)
        test_prj.net.add_link(3,4)
        test_prj.net.add_link(4,5)
        test_prj.net.add_link(5,1)
        test_prj.net.add_link(6,7)
        test_prj.net.add_link(7,8)
        test_prj.net.add_link(8,1)
        test_prj.net.add_link(4,8)

        # simplify project
        print 'Original project:'
        test_prj.print_me(1)
        test_prj.simulate(50000)
        test_prj.sdur.write_me('ProjSim')
        test_prj.calc_dur()
        print 'Simplified project:'
        test_prj.print_me(2)
        test_prj.cdur.write_me('ProjCalc')

    else:
        timeA=U_time()
        timeB=U_time()
        timeA.triangle(4,5,7,0.1)
        timeB.triangle(5,6,8,0.1)
        histA=[]
        histB=[]
        for val in timeA.pdf: histA.append(0.0)
        for val in timeB.pdf: histB.append(0.0)

        iter=100000
        i=0
        while i<iter:
            valA=timeA.simulate()
            indA=int((valA-timeA.min+0.05)/timeA.prec)
            histA[indA]=histA[indA]+1.0/float(iter)
            valB=timeB.simulate()
            indB=int((valB-timeB.min+0.05)/timeB.prec)
```

```
            histB[indB]=histB[indB]+1.0/float(iter)
            i=i+1
        outfile=open('HistA.csv','w')
        dur=timeA.min
        for val in histA:
            tempstr='%8.4f'%dur
            tempstr=tempstr+','+'%12.8f'%val+'\n'
            outfile.write(tempstr)
            dur=dur+timeA.prec
        outfile.close()
        print 'File HistA.csv written and closed'
        outfile=open('HistB.csv','w')
        dur=timeB.min
        for val in histB:
            tempstr='%8.4f'%dur
            tempstr=tempstr+','+'%12.8f'%val+'\n'
            outfile.write(tempstr)
            dur=dur+timeB.prec
        outfile.close()
        print 'File HistB.csv written and closed'

        timeA.write_me('TimeA')
        timeB.write_me('TimeB')


# Program startup
if __name__ == '__main__':
  main()
```