

Research Notes for Chapter 16*

Chapters 9 and 10 of *Introduction to Sequencing and Scheduling* (Baker 1974) address project scheduling. The former covers fundamental concepts and is the basis of our Chapter 16; the latter covers sequencing activities that compete for resources and is the basis of our Chapter 17. However, that coverage was not included in *Elements of Sequencing and Scheduling* (Baker 2005) because of the book's focus on deterministic machine scheduling models. We reintroduce the project scheduling chapters not only because the current text includes stochastic models but also because project models have become more important and because safe scheduling has been developed primarily in the context of project scheduling. Notably, 35 years after the publication of *ISS*, we found precious little to update in this chapter. We might expect that the fundamentals should not change with age, but it is surprising that the same criticisms noted then remain valid today. Moreover, most of those criticisms have not been properly addressed until very recently, if at all. The bulk of useful new work has focused on deterministic (CPM) sequencing models, so our Chapter 17 is quite different from Chapter 10 of *ISS*. We address the most recent safe scheduling developments in Chapter 19 and its Research Notes. Together, those three chapters may be viewed as a module.

Below we provide the main sources on which Chapter 9 of *ISS* and the current chapter rely. We also introduce additional material. Specifically, we discuss some technical issues including network structure, the distribution of project length, crashing, hierarchical project management, and—last, but not least—a very recent, radically revised PERT/CPM framework which we call *PERT 21* (or *21st Century PERT*). PERT 21 is designed to combine the useful parts of both PERT and CPM and provides a fertile new direction for research in project scheduling.

Sources and Comments

According to Kelley (1961), the earliest paper on CPM is Kelley (1957) but the complete structure is attributed to Kelley and Walker (1959) and to a 1958 working paper by the same authors. Kelley (1961) is self-sufficient and more accessible than any of those earlier sources. Furthermore, it also includes a solution of the crashing linear program by a network flow model.[†] In a nutshell, that network flow model is the dual of the model we presented in Section 16.4. A similar solution was independently developed by Fulkerson (1961) and submitted for publication in the same month (June 1960). Kelley and Fulkerson influenced each other's work. Kelley uses a network flow model developed by Ford and

* The Research Notes series (copyright © 2009, 2019 by Kenneth R. Baker and Dan Trietsch) accompanies our textbook *Principles of Sequencing and Scheduling*, Wiley (2009, 2019). The main purposes of the Research Notes series are to provide historical details about the development of sequencing and scheduling theory, expand the book's coverage for advanced readers, provide links to other relevant research, and identify important challenges and emerging research areas. Our coverage may be updated on an ongoing basis. We invite comments and corrections.

Citation details: Baker, K.R. and D. Trietsch (2019) Research Notes for Chapter 16 in *Principles of Sequencing and Scheduling* (Wiley, 2019). URL: <http://faculty.tuck.dartmouth.edu/principles-sequencing-scheduling/>.

[†] Network flow models are linear programs with a special structure that allows a much more streamlined solution than the general simplex method.

Fulkerson in the late 1950s (see Ford and Fulkerson, 1962), whereas Fulkerson solves a problem first posed by Kelley and Walker (1959). Kelley (1963) and Wiest (1964, 1967) address the issue of sequencing when resource limitations prohibit as much processing in parallel as the basic network diagram would permit. We discuss such sequencing models in Chapter 17. More complex network structures have also been developed. One, GERT (where G stands for *generalized*), recognizes that when a project is executed, some activities may be skipped, based on the outcome of other activities, so the network structure itself is random (Elmaghraby 1972). For example, in R&D projects it may make sense to develop several new designs in parallel until one of them proves adequate. Each design may require a potentially different set of downstream activities, so we cannot tell in advance which subsequent activities will be undertaken. Similarly, earlier developments may have to be reworked based on downstream events, so the project may include a corrective loop. Because activity networks cannot include loops, this structure has to be depicted as a project that may duplicate activities downstream. More generally, the number of times we go through such a loop is random and each possible realization implies a different network. The assumption is, however, that we can assign probabilities to each realization, so nodes may branch to different activities with given probabilities. Nodes may also be activated in more elaborate ways than in PERT. For instance, a node may be activated upon completion of the first activity that leads to it (rather than upon completion of *all* activities that lead to it). Another generalization addresses the precedence relationships between activities. In our text we discuss the simplest type, where a predecessor activity must finish before the successor can start, which we can address as *finish-start*. Other options include *start-start* (an activity cannot start before another activity starts), *finish-finish* (an activity cannot finish until another activity does), or *start-finish* (an activity cannot finish before another activity starts). For a detailed discussion of such network structures, see Demeulemeester and Herroelen (2002). Among these enhancements, GERT addresses an important practical concern—random networks abound in practice—whereas the other structures are mainly of academic interest. At present, combining GERT with considerations of safe scheduling would represent a promising area of research.

The development of PERT for the US Navy started sometime after the publication of Kelley (1957) and yielded technical reports in 1958 and a refereed publication a year later, written by the leading members of the PERT team (Malcolm et al. 1959). Our statement in the chapter that CPM and PERT were developed independently is based on Kelley (1961), who states that the models were developed "in parallel." Important follow-up papers include Van Slyke (1963) and MacCrimmon and Ryavec (1964). Van Slyke introduced the use of simulation for stochastic project scheduling, including criticality calculations. The term *criticality*, however, is attributed by Van Slyke to MacCrimmon and Ryavec (even though their paper appeared later), and our Example 16.5 is drawn from their work.

Whereas the citations above are early sources of PERT/CPM theory or early enhancements thereof, an important body of work focuses on identifying PERT deficiencies or on correcting them, as discussed in the chapter. To a large extent, this criticism applies to PERT as implemented rather than to PERT as it was intended. Malcolm et al. (1959) is presented as a case study, describing the development of the model and its implementation. During that process, time was of the essence and some shortcuts and

approximations were deemed necessary. However, the authors do not suggest that the same shortcuts and approximations should necessarily become standard (although, in fact, they did). Given that background, it may not be surprising that the first critique of the PERT method appears in the original paper. Specifically, the authors note that equating the critical path with the maximal mean path is just a convenient approximation. Essentially, they recognize that this calculation ignores the Jensen gap. They state that a better method already existed, and indeed an improved calculation that approximately accounts for the Jensen gap was subsequently published by Clark (1961). Clark's paper should have become a well-known classic, but for a long time it attracted very little attention.* Clark still assumes activity times are independent, but his approximation does not really require that assumption.

Malcolm et al. also cite common practitioner concerns that the PERT activity estimation method could lead to bias; that is, these practitioners did not believe that experts could reliably provide accurate estimates.† To respond to the bias issue, they recommend a program of calibration. That recommendation has never, to our knowledge, been followed.‡ Much more attention was devoted to the Jensen gap issue, where two types of response emerged. One approach—of which Clark (1961) is the earliest exemplar—employs approximations or bounds to estimate the true mean. Another early contribution along these lines is Fulkerson (1962), who assumes discrete activity distributions and calculates a bound on the mean. Fulkerson actually allows dependence between the activities that precede any single node but requires independence among activities that precede different nodes. Thus, in practice, his approach can only model very special cases of dependence and requires replacing continuous random variables by approximate discrete ones. Another important paper that uses bounds, but without requiring discrete distributions, is Dodin (1985). Subsequent papers propose ever more complex but tighter bounds. The other approach, and the one we recommend in our text, is simulation (Van Slyke, 1963). Again, as in all the well-known papers about the subject, Van Slyke assumes independence, although simulation models do not really require the independence assumption. For some reason, the issue of correlation between activities was simply glossed over, until recently. Furthermore, the Jensen gap is implicitly assumed to be the main cause of problems associated with PERT. That view is represented in several papers, including Klingel (1966) and Schonberger (1981).

As noted in the chapter, PERT did not require the beta assumption to begin with. Indeed, Clark (1962) states explicitly that it was selected quite arbitrarily and that "[t]he author has no information concerning distributions of activity times, in particular, it is not

* Klingel (1966), one of the early papers that does cite Clark, may have contributed to that state of affairs by suggesting that Clark's approximation had not been tested, but in fact Clark provides extensive results showing the performance of his algorithm under various assumptions. Klingel also states that Clark's approximation is computationally intensive, which would certainly be a valid concern for manual calculations but not an issue for a computerized application (even in those days, especially when compared to Klingel's own use of simulation). Incidentally, as stated in our Research Notes for Chapter 11, we rely on Clark (1961) in Equation (11.4). The detailed derivation of that result is covered in Appendix B.

† Using standard probabilistic terminology, an expert provides *accurate* estimates if he or she is on target *on average*. Otherwise, the estimates are *biased*. Accuracy should be distinguished from *precision*, however, which is a measure of variation: a precise estimate, when corrected for bias (if any), is exactly on target, but without correcting for bias it could be consistently off-target.

‡ Trietsch et al. (2010) demonstrate that following that recommendation would have been beneficial but not sufficient.

suggested that the beta or any other distribution is appropriate." To repeat: the selection of the beta distribution was *not* based on any claim that it is inherently likely to be the best fit. Nor are Equations (16.3) and (16.4) based on empirical evidence that they are appropriate. Instead, they had been selected quite arbitrarily: (16.4) was subjectively judged to be appropriate and (16.3) was then specified as a sufficiently good approximation (Clark, 1962). The main practical problem caused by this assumption may be that the variance estimate (and thus also the coefficient of variation) is likely to be exceeded in practice (Trietsch et al., 2010). In other words, (16.4) is not only arbitrary but also tends to be highly optimistic. On a technical front, the assumption motivated a lot of effort devoted to improved estimation of the beta parameters. The technical problem is that Equations (16.3) and (16.4) do not precisely fit *any* beta distribution except for the symmetric case where $m = (a + b)/2$ and two other special cases pointed out by Grubbs (1962). With the exception of those three special cases, if (16.3) is correct, (16.4) is not, and vice versa. Grubbs goes on to demolish the PERT estimation method on both technical and practical grounds. Unfortunately, Grubbs' criticism was not heeded, perhaps due to a reason he anticipated when he stated that "it is not with any great satisfaction that the reviewer can accept statements to the effect that 'irrespective of your theoretical comments, you are picking on small points since PERT works so well.'" In other words, Grubbs warned against interpreting the success of managing the Polaris project by PERT as conclusive evidence that PERT is completely meritorious in general and that the use of the beta distribution is necessarily correct in particular.* Whereas Grubbs questioned the inherent validity of the PERT estimation method, thus calling for a new model altogether, most of the subsequent literature on the subject offers technical corrections and clarifications without questioning the basic structure again. Sasieni (1986) shows that by holding the sum of the two beta parameters to a particular constant (namely 4), Equation (16.3) is satisfied. However, he ignores the fact that Equation (16.4) is no longer valid in that case (with the exception of the three special cases listed by Grubbs). Several authors tried to present better approximations for the beta parameters. One approach is to keep (16.4) intact but to replace (16.3) by another calculation for μ that will match some beta distribution precisely. In general, even that limited objective cannot be achieved precisely for the whole range of possible mode locations (between the min and the max), but it can be done for a wide range. For a relatively sophisticated approximation of that type, plus a review of earlier approximations, see Premachandra (2001). Our own perspective is that the lognormal distribution is simply better suited to the model than the beta. This claim is based not only on the theoretical arguments provided in our text but also on field data, as we discuss later.

The Relationship between AOA and AON Networks

The AOA network depiction is often criticized because it requires dummy activities, whereas AON networks appear to avoid that need. However, as observed by both Kelley (1961) and Fulkerson (1961), both can be generalized to a single AOA depiction where every single arc from the original AON network becomes a dummy in the generalized version (Figure RN16.1). Start with an AON network where node j represents the j th

* Misinterpreting commercial or practical success as theoretical validation has been the root cause of quite a few historical aberrations. Serious practitioners and scholars are well advised to beware that trap. Instead, practical success of any method should motivate objective scientific study. The operative word, however, is *objective*.

activity. Replace the original node j by two nodes, j' and j'' , and connect j' to j'' by an arc. Redirect all the original incoming arcs into j' and start all outgoing arcs from j'' . The new arc represents the activity, and j' and j'' are its start and finish events, so we obtain an equivalent AOA network. Furthermore, because all original arcs take zero time and resources, they are all dummies. Accordingly, we may replace them by dashed arcs. At this stage it is possible to *collapse* some of these dummy activities by merging their end nodes. This can be done for any dummy activity that is not necessary to satisfy rule 5 (that no two activities should share more than one end node). We now see that the AOA network shows events more explicitly, whereas the AON network shows logical constraints more explicitly. The network obtained by the construction we described, before collapsing any dummy activities, is the most explicit depiction. Accordingly, we refer to it as the *explicit network*.

The process we described is also a legitimate way to transform AON networks to AOA networks. Its advantage is that alternative dummy placements become visible, so we can keep the ones we prefer and collapse the others. Moreover, logical construction errors are less likely. To illustrate, return to Example 16.1. Figure RN16.1 shows the transformation of the AON network of Figure 16.6, on the left, to the explicit network. This explicit network contains four dummy activities. We can collapse any three dummy activities and obtain a legitimate AOA network for the example. Figure 16.5—repeated below as Figure RN16.2—reflects one of these four choices, where we kept the dashed arc from B to D and collapsed all other dummies.

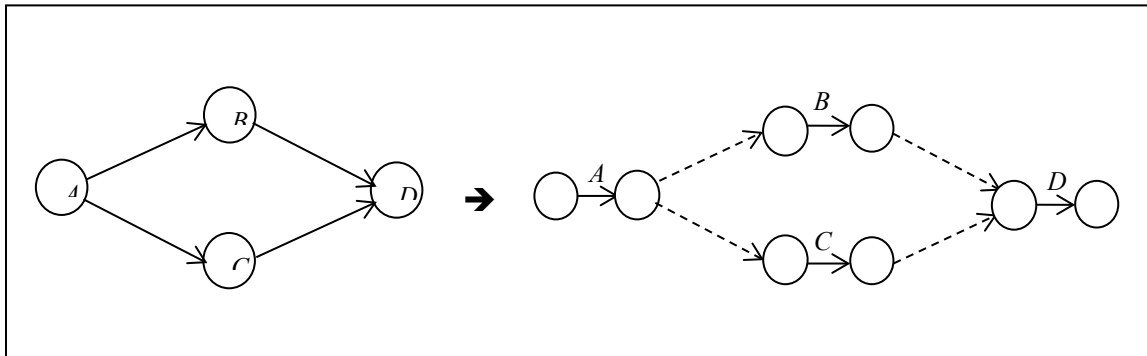


Figure RN16.1. Transforming an AON Network to an Explicit Network.

Incidentally, in an AOA network it may happen that all successors of an activity are dummies. Notably, this may happen even if no dummy is redundant; e.g., see the successors of activity A in Figure 18.2. In such case, the free-float calculations should be slightly revised by adding the minimal free float assigned to such a dummy successor to the activity itself (Zhao and Tseng 2003).

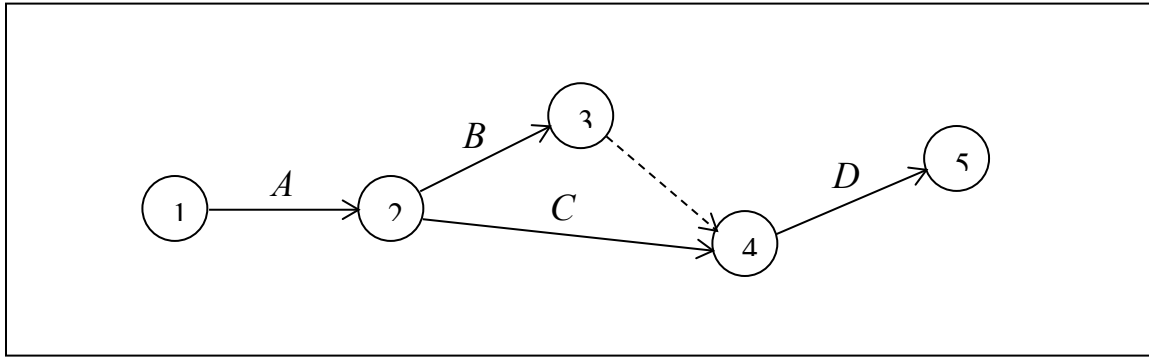


Figure RN16.2. An AOA Network for Figure RN16.1.

Series-Parallel Project Networks

In Chapter 8 we described series-parallel networks. Such networks may also describe projects. For instance, take Example 8.4, which we presented by an AON network in Figure 8.3, repeated here as Figure RN16.3. The same network could also describe a project. Figure RN16.4 is an AOA depiction of that project, and we focus on its makespan—that is, the completion time of the last node. Because the project is series-parallel, it is possible to compute the cumulative distribution function (cdf) of its completion time without resorting to simulation, rough approximations or bounds, provided all activities start at their ES time. This process is called *reduction*. Generally, it is assumed that all processing times are independent, but we now show that reductions can also be carried out for series-parallel networks when activity times are linearly associated.

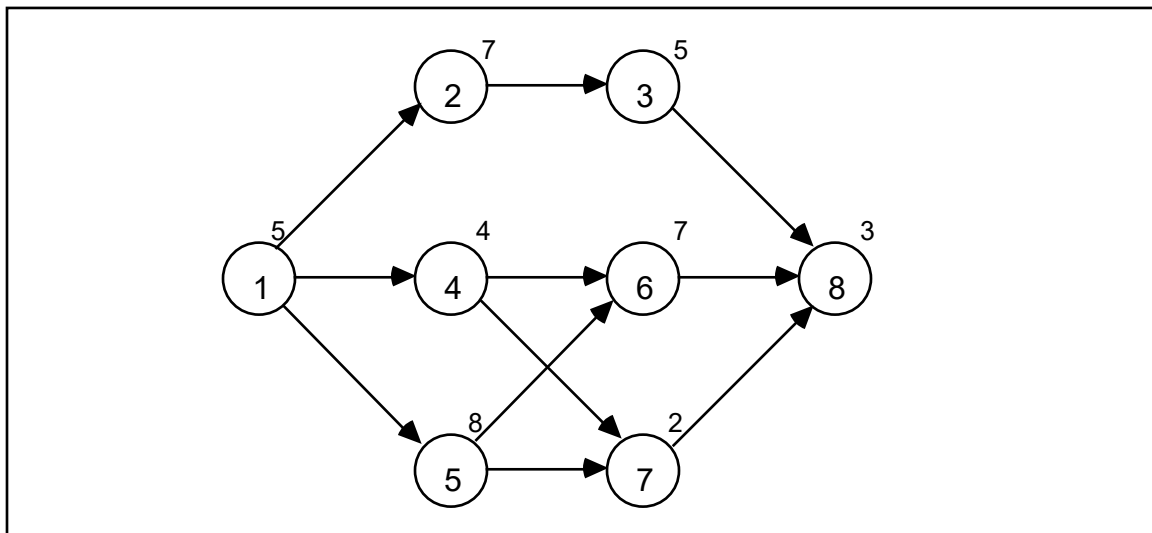


Figure RN16.3. An AON, 8-Activity Project (after Figure 8.3).

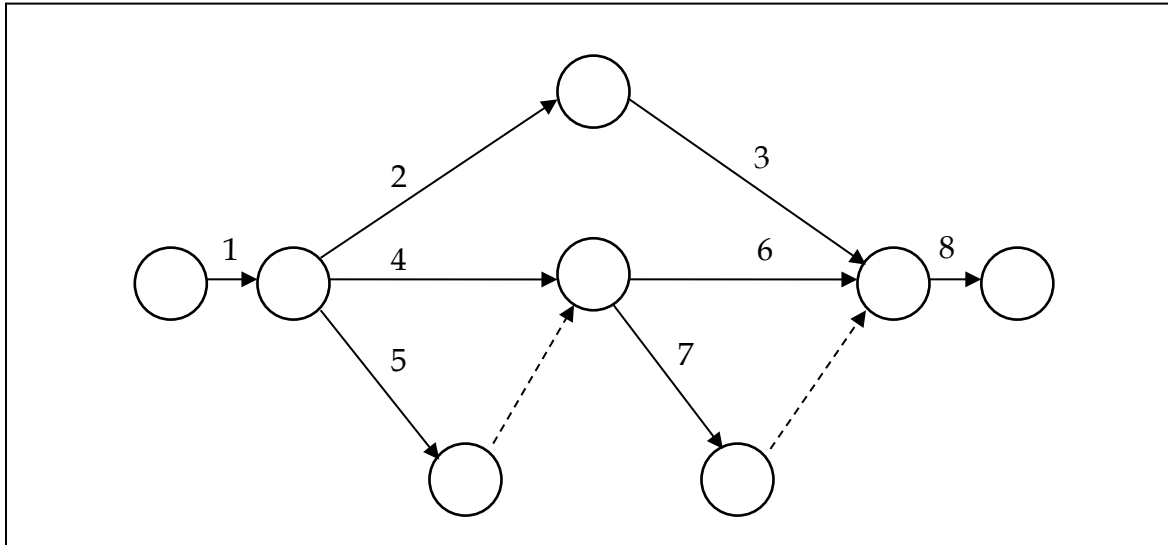


Figure RN16.4. An AOA Depiction of Figure RN16.3.

Temporarily, assume that a network is series-parallel with a given decomposition tree. We first describe the procedure for independent activities. Initially, the leaves of the decomposition tree (nodes without successors) all represent activities, and we associate the cdf of the relevant processing time with each of them. As we progress, we collapse subsets of leaves into their parent nodes, which become new leaves with cdfs that represent the composite effect of all leaves collapsed into them. Consider any set of leaves in the decomposition tree that have a common parent node. If the parent node is S (series), replace the cdfs of the leaves by the cdf of their convolution, and associate that cdf with the parent node (which becomes a leaf). Convolutions are often impossible to obtain analytically but can always be computed by numerical methods. If the parent node is P (parallel), replace the cdfs of the leaves by the cdf of their maximum, obtained by their product (which, again, can always be obtained by numerical methods). When the tree is reduced to just one leaf, the cdf of that leaf is the cdf of the whole project. For instance, if we consider Example 8.5, the decomposition tree is given in Figure 8.4, repeated as Figure RN16.5. To find the cdf of the makespan, working from right to left and top to bottom, we start by representing the cdfs of leaves 6 and 7 by their product at the P node. Similarly, we represent the cdfs of leaves 4 and 5 by their product. The two new cdfs are now connected to an S node, so they are represented by the cdf of their convolution. Leaves 2 and 3 are also replaced by a convolution at its S parent node. The next node is P, so the two new cdfs are multiplied to get the maximum. That maximum is convoluted with 1 and subsequently with 8, to obtain the final cdf of the makespan. It is also possible, however, to perform reductions without an explicit tree. In fact, one way to identify series-parallel networks is to try to reduce the project by iteratively convoluting all serial structures and replacing all parallel activities cdfs by the cdf of their maximum.

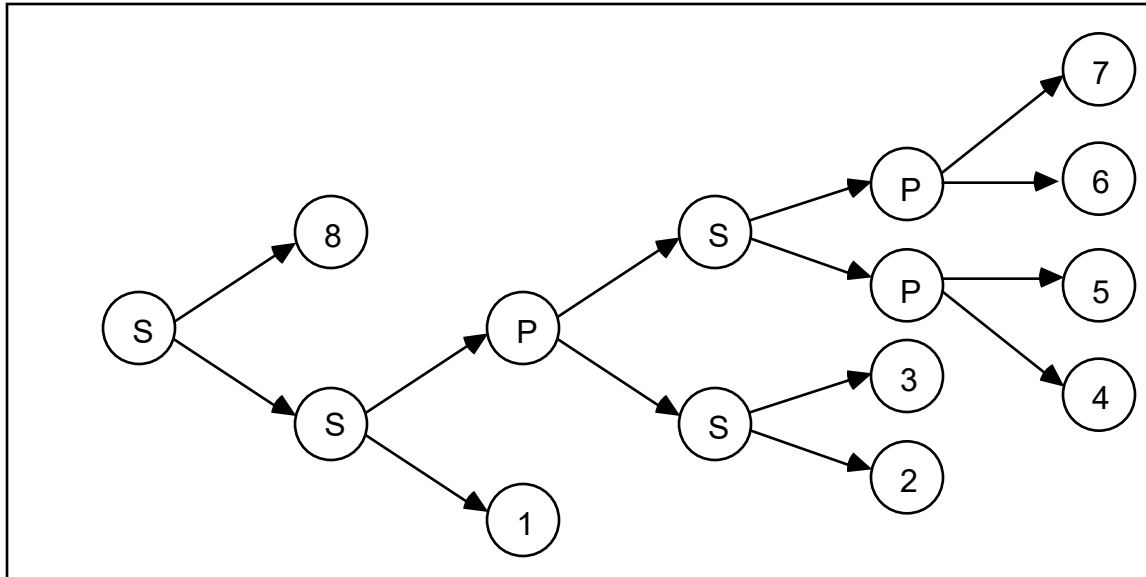


Figure RN16.5. A decomposition tree for the example in Figures RN16.3 and RN16.4.

This process can be carried out in any order. Using the AOA depiction, the condition for convolutions is that two activities can be convoluted only if the finish-node of one coincides with the start-node of the other and no other activity starts or ends at the same node. The convolution effectively removes that node, and for that purpose we allow two arcs in the reduced network to start and end at the same nodes (thus violating rule 5). For instance, in Figure RN16.4, activities 2 and 3 can be convoluted because no other activities start or end at the node between them. We can also see this result in the decomposition tree, where 2 and 3 are connected to an S node. Similarly, the only condition for performing the maximum operation on two parallel activities is that if we ignore dummies, they share the start node and the end node (again, we relax rule 5). For instance, in Figure RN16.4, activities 4 and 5 (and, likewise, activities 6 and 7) can be replaced by the cdf of their maximum, because if we collapse the dummy into activity 5 the two remaining activities (4 and the new 5) start and end at the same node. As the network evolves due to such operations, we continue to pursue emerging possible operations with the objective of reducing the network to a single cdf. Upon success, we can say that the network is series-parallel. This process is equivalent to the one we described with the decomposition tree, but the decomposition tree is implicit. However, if we encounter the interdictive graph during the process, we cannot continue, the network is not series-parallel, and no decomposition tree exists. To wit, in the interdictive graph (see Figure 16.16 which is also repeated below as Figure RN16.6) there are no two activities in series where the common node is not used by any other activity and there are no parallel activities, either.

For linearly associated activities, we start by solving for the initial independent distributions (see Appendix A). Next, to get the adjusted distribution we account for the common factor, Q . Technically, that operation is similar to a convolution and requires integration. Let F and f represent the cdf and density function of the unadjusted completion time, let G and g represent the common bias, and let H and h represent the product. Because both components are strictly positive, we have

$$H(z) = \int_0^z f(x)G\left(\frac{z}{x}\right)dx = \int_0^z g(x)F\left(\frac{z}{x}\right)dx$$

In a convolution, instead of $G(z/x)$ or $F(z/x)$, we have $G(z-x)$ or $F(z-x)$, respectively. The validity of this approach follows by the next theorem, which is essentially a rewording of Theorem A.4 in terms of project scheduling.

Theorem RN16.1: Consider a project where all activities are available at their early-start time (i.e., without active release dates). Assume linearly-associated processing times with a common factor element B . Let C_j be the completion time of activity j , and let C_j' be the completion time that would apply without the common factor. Then $C_j = BC_j'$.

Parametric Crashing and the Network Flow Model

In the chapter we presented a linear programming crashing model that minimizes the total crashing cost for a given fixed periodic cost c_f . A *parametric* version of the same problem allows c_f to vary, as a parameter. Alternatively, we can set the makespan to any desired constant, λ , and find the cheapest way to achieve it. This alternative approach was adopted by Kelley (1961) and by Fulkerson (1961). To solve both parametric problems we must find a set of efficient crashing plans such that one of them is optimal for any given c_f . Just identifying that set of *basic solutions* is sufficient for the first version because one of the basic solutions is optimal for any c_f . To solve the version presented by Kelley and Fulkerson, however, we need an additional step because the required duration, λ , may not be achieved precisely by any basic solution: If λ is smaller than the fully crashed makespan, there is no feasible solution. If it is larger than the makespan without crashing, the optimal solution requires no crashing but we may start the project at its latest start time (with λ acting as a due date). If λ is within the range of durations of the basic solutions, we use an appropriate weighted combination of the two adjacent basic solutions above and below λ .

As it happens, the steps we went through in our heuristic solution of Example 16.3 provide the required set of basic solutions. For a sufficiently small c_f , up to 200, no crashing is the best option. For $c_f > 200$, it becomes cost-effective to crash activity D by two days (we can also say that for $c_f \geq 200$, because if $c_f = 200$ we lose nothing by that crashing plan). The plan remains optimal for $200 \leq c_f \leq 300$, but for $c_f > 300$ we should crash D by an additional day and crash B by one day in parallel. That plan remains optimal for $300 \leq c_f \leq 400$. For $400 \leq c_f \leq 500$, it is optimal to also crash A by 2 days (which is the solution of Example 16.3 because we had $c_f = \$450$), and for $c_f > 500$ we should also crash B by an additional two days in parallel to activity C . In this case, the solution procedure of the example essentially solves the parametric problem as well. We note that the procedure involves identifying the best crashing plan *given* the previous plan in such a manner that no decision is ever reversed. Thus, as we progress to higher c_f values, we look for additional crashing opportunities without ever reversing a previous one. It can be shown that this is the case because the network of Example 16.3 is series-parallel. When the network is not

series-parallel, as we increase c_j , it may become necessary to reverse previous crashing decisions.

Example RN16.1. Consider a project with the network of Figure RN16.6 (which is the simplest non-series-parallel network possible). Crashing information is given below:

<i>Activity ID</i>	<i>Predecessors</i>	a_j	b_j	c_j
<i>A</i>	—	3	5 days	\$11
<i>B</i>	—	7	9	10
<i>C</i>	<i>A</i>	3	5	10
<i>D</i>	<i>A</i>	7	9	10
<i>E</i>	<i>B, C</i>	3	5	11

The initial critical path is $\{A, C, E\}$, with a duration of 15. Because C has the lowest crashing cost within the critical path, we crash it by one day to 4 days. At this stage, the critical path is reduced to 14 days, and the total crashing cost is 10. Every activity is now critical. Following the logic of Example 16.3, one might be tempted to crash activities A and B , or symmetrically, activities D and E , at a cost of 21 per day in both cases. However, if we crash A and E , the marginal cost per day is *not* 22 but 12. The reason is that we can reverse the previous crashing decision of activity C and save 10. That is feasible for just one day (because we crashed C by only one day) and leads us to a project duration of 13 with a total crashing cost of $10 + 12 = 22$. Next we can crash activities A and B , or symmetrically, activities D and E , as mentioned before. Because A and D are already partly crashed, each of the two options offers a marginal duration reduction of one day. Together they lead to a duration of 11 days at a cost of $10 + 12 + 21 + 21 = 64$. We can now crash activities B , C , and D by one day, at a marginal cost of 30; the minimal feasible duration is thus 10 and the total crashing cost is $64 + 30 = 94$. During the process, activity C was tentatively crashed, de-crashed, and re-crashed.

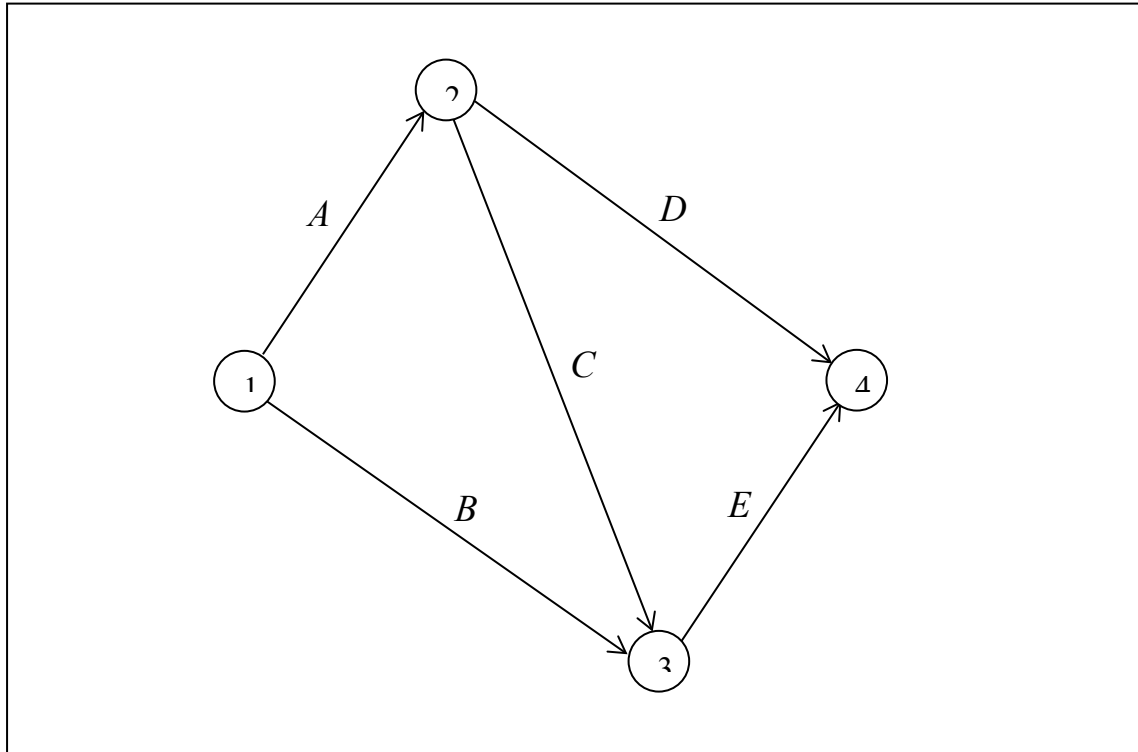


Figure RN16.6. The Interdictive Graph

In our solutions of both Examples 16.3 and RN16.1, we identified the best crashing opportunities by essentially comparing all options. Both Kelley (1961) and Fulkerson (1961) observe that it is possible to use the network flow model to identify these opportunities efficiently. The technical details of the Ford and Fulkerson network flow algorithm are not complex, but they are beyond our scope. Actually, in a practical sense, they are less important today than they were at the time because modern personal computers can solve the basic crashing LP for large projects without much difficulty by using generic LP solvers, so there is no compelling practical reason to utilize the network flow algorithm in this application. Interested readers can find the versions used by Kelley and by Fulkerson in their respective 1961 papers, and the classic reference is Ford and Fulkerson (1962). But it is instructive to study the underlying structure that makes this approach applicable.

One of the most insightful results developed by Ford and Fulkerson is known as the min-cut max-flow theorem [minimum-cut = maximum-flow]—essentially a duality result that identifies the set of connections in a network that limit its throughput, thus acting together as the bottleneck. To explain the relationship to crashing, we need to define cuts in our context. We have already used cuts informally in our solutions of Examples 16.3 and RN16.1, but now we provide a slightly more formal description. Consider any AOA directed activity network with a start node and a finish node, to which we also refer as

source and *sink*.^{*} Suppose there are N nodes, then there are 2^{N-2} ways to partition them to two complementary subsets with one including the source and the other, the sink. Our interest is confined to a subset of these partitions, called *proper*. To define proper partitions, we start by distinguishing between directed paths and [unspecified] paths. The definition of a directed path is recursive. There is a directed path between two distinct nodes i and j if activity (i, j) exists (in which case the path is also *basic*), or if for some node k there is a directed path between i and k and a directed path between k and j . The critical path is an example of a directed path between the start and finish nodes. Such a directed path can be denoted by the ordered set of all nodes which it traverses; for instance, when activities (i, k) and (k, j) form the directed path, it is denoted by $\{i, k, j\}$. By our rules (as given in the chapter), if a directed path $\{i, \dots, k, \dots, j\}$ exists then $i < k < j$. A not-necessarily-directed *path* is defined similarly but without regard to orientations. A *basic path* connects two distinct nodes, i and j , if either activity (i, j) or activity (j, i) exists. More generally, a path connects i and j either if they are connected by a basic path or if, for some distinct node k , there is a path between i and k and a path between k and j . By definition, directed paths are a subset of paths. For example, consider Figure RN16.6. There is a path through nodes $\{1, 2, 3\}$ which is directed, and there is a path through nodes $\{1, 3, 2\}$ which is not directed (and indeed 3 appears before 2 in the ordered set but $(3, 2)$ cannot be a legitimate activity because $2 < 3$). Considering nodes 1 and 2, there are one (basic) directed path and two additional undirected paths connecting them (namely $\{1, 2\}$, $\{1, 3, 2\}$, and $\{1, 3, 4, 2\}$). We say that a subset is *connected* if any two nodes in the subset are connected by at least one path that does not visit any node outside the subset. For a partition to be proper, we require both subsets defined by it to be connected. To explore that notion further, we define a *cut set* as the set of all basic paths (in our context, project activities) that start in one subset and end in the other. For instance, let the first subset consist of nodes 1 and 3, and we note that there is a direct path connecting 1 to 3 and another direct path connecting 2 to 4, so the partition is proper. For these subsets the cut set consists of $(1, 2)$, $(2, 3)$, and $(3, 4)$. If we cut these arcs (that is, delete them from the project network), then there will be no path connecting the source to the sink; hence the term "cut set." Now suppose that the cut set for a given partition leads to more than two connected subsets. This situation can occur if, not counting the source, a node in the first subset has no direct predecessor in the first subset. A symmetric case occurs if a node in the second subset, not counting the sink, has no direct successor in the second subset. In the former case, all direct predecessors of a node in the first subset are in the second subset and there can be no chain connecting it to the first subset that does not include nodes from the second subset. Hence, the first subset is not connected. Using the cut set for a partition with such structure, once we cut all the connections of the node to the second subset, it is not connected to either subset. But in a proper partition there are two connected subsets and every node must belong to one of them. In the symmetric case, the second subset is not connected. When a partition is proper, the cut set associated with it is also called *proper*. One characteristic of a proper cut set is that if we fail to cut any of the arcs in it, the source and the sink remain connected. In other words, when the objective is to cut the source from the sink, a proper cut set does not include any superfluous arc.

^{*} Kelley (1961) calls them *origin* and *terminus*, but the name *terminal* is often used as well. The terms *source* and *sink* follow the notation favored by Fulkerson, and suggests a focus on flows, as in the network flow model.

At any stage in a project's course, we can identify a subset of nodes (including the source) which have been completed already and the complementary subset of nodes that have not yet been completed (including the sink). This defines a proper partition. Furthermore, when we consider the cut set of this partition, we can see that all activities on it start in the first subset and end in the second subset. Such cuts, called *unidirectional cuts* (UDC), are a subset of the proper cuts. The UDCs in our example are $\{(1, 2), (1,3)\}$, $\{(1, 3), (2,3), (2, 4)\}$, and $\{(2, 4), (3,4)\}$ (or, equivalently, $\{A, B\}$, $\{B, C, D\}$ and $\{D, E\}$), whereas the cut set $\{(1, 2), (2, 3), (3, 4)\}$ ($\{A, C, E\}$) is proper but not unidirectional (C starts in the second subset and ends in the first subset). By definition, at any given time only the activities of some UDC can be performed in parallel. When one activity is complete, the UDC may or may not change, depending on whether *all* activities with the same ending node are also complete. A proper cut must include at least one arc directed from the first subset to the second. Such arcs are *positively directed*. A UDC comprises only positively directed arcs. Other proper cuts also include *reversed* arcs, which are directed from the second subset to the first. For example, in the cut $\{A, C, E\}$, C is a reversed arc. Suppose that each arc has a capacity to support a flow, and we wish to maximize the total flow from the source to the sink through the arcs without violating the capacity of any arc and such that if we exclude the source and sink, flow is conserved—i.e., all the flow coming into a node must exit it. The source is allowed to generate a net positive flow and the sink absorbs it. (Conceptually, if we connect the sink to the source by another arc, we obtain circular flow—which we have not allowed in our analysis so far—the flow is conserved in these nodes as well. This is often done for flow analysis.) The capacity of an arc is an upper bound on the flow assigned to it, but in general, lower bounds may also apply (e.g., in cold weather a pipe that carries no flow may freeze). For our purpose, we can safely ignore such lower bounds. For this case, the *capacity* of a cut is the sum of the capacities of the positively directed arcs in it. (In the presence of minimal flow constraints, the capacity of a cut is reduced by the sum of the lower bounds on the reverse arcs.) The min-cut max-flow theorem states that the maximal flow possible is equal to the capacity of the minimal cut (the *bottleneck*). It is clear that the maximal flow cannot exceed this capacity, because the flows must cross the cut and there is no way to support more than the total capacity of the arcs in any cut, including the minimal one. The non-trivial part of the Ford and Fulkerson proof essentially shows that it is also *possible* to allocate flows to arcs in such a way that the capacity of the minimal cut will be fully utilized. (This is perhaps the earliest capacity bottleneck result in the literature.)

The duality between the min-cut and the max flow forms the connection between the network flow problem and crashing. The best crashing plan involves selecting the best cut, but by duality there is an associated maximal flow associated with it. In more detail, consider any cut, and clearly at least one activity in the cut must be critical (because the critical path must cross the cut). Now associate with each activity that is critical but not yet fully crashed a crashing cost of c_j . An activity that has positive float is allocated a crashing cost of zero (i.e., we treat the removal of float as a form of crashing), and a fully crashed activity gets a crashing cost of ∞ (or a sufficiently high positive value that will prevent any attempt to crash it further). However, if a reversed arc is crashed, it gets a crashing cost of $-c_j$ (because we crash such an activity by reducing the original amount by which it had been tentatively crashed). If we decide to crash a set of activities that form a cut, then the cost of crashing the project by one time unit is the sum of the crashing costs on that cut.

Thus, the optimal way to crash the project by one time unit is to find the cut that has the minimal total crashing cost. That is exactly what we did in Example 16.3, but we only had UDCs to contend with, which simplifies the analysis. In Example RN16.1, the cut $\{A, C, E\}$ was not unidirectional. But to solve the parametric crashing problem we had to use this particular cut in the second step. In the general case involving reversed arcs, if such a reversed arc has been tentatively crashed in a previous stage, and it later forms part of a minimal cut, we can reduce the amount of crashing on that arc and recoup the crashing cost that was tentatively allocated to it; for this reason we assign it a negative crashing cost. If no previous crashing of that sort exists on such an arc and we crash the other activities on the cut, we add float to this arc. Thus, if it is possible to reverse a previous crashing decision on a reversed arc we can reduce the crashing cost of that cut and this would make the cut more attractive, but otherwise such arcs can be ignored. So what we need to do is to find the cheapest cut and if any previously crashed arc in a cut is reversed we should take into account that it now becomes possible to reverse that decision. Reiterating the connection to network flow, if the minimal cut defines the maximal flow, it should be possible to form the problem of identifying and using the minimal cuts as a network-flow model because there is a conceptual maximal flow associated with this minimal cut. Essentially, similarly to the manual solution we presented for Examples 16.3 and RN16.1, the idea is to identify the current cheapest cut and crash it maximally if it is cost-effective. Once any arc on that cut is fully crashed, the cut is no longer available, and we seek the next cheapest cut (which may involve reversing one of the previous crashing decisions), etc. However, if the network is series-parallel, all proper cuts are UDCs, and it is never useful to reverse a tentative crashing decision.

On Hierarchy

Although Malcolm et al. (1959) discuss the possibility of representing subprojects by composite activities, they explicitly state that a project network—such as the one used for the Polaris project—can include thousands of activities. Indeed, that is the way PERT has been used in similar projects. The resulting network drawing can easily be very large. For instance, one of us witnessed a PERT network for a ship overhaul project in the US Navy that measured about 20 feet long by 2 feet wide, with quite fine print. Such networks have so much detail that it is almost impossible to comprehend the full picture. By contrast, CPM was designed differently. Kelley (1961) states that each activity can and often should represent a subproject, so the total number of activities on a chart does not become excessive. (That representation also has implications for the crashing model. Essentially, it implies that the unit crashing cost of an activity actually represents the optimal crashing cost of a subproject.) Kelley's recommendation amounts to a hierarchical representation of projects. This approach yields networks of more modest sizes—say 50 to 300 activities—that are easier to understand. Moreover, given the limited computer capacities available when Kelley was writing, such networks would be more amenable to an optimization approach than a network containing the full detail. Even with today's more powerful computers, this approach still has merit when we consider the complexities of stochastic analysis and sequencing (which neither Kelley nor the PERT team addressed at the time). Because PERT aimed to coordinate thousands of activities, it did not adopt an optimization approach. Instead, it was designed to help managers make time/cost and resource allocation decisions by asking "what if?" questions—that is, PERT was primarily a decision support

system that did not optimize decisions but helped managers assess the effect of possible options. By contrast, CPM took a deterministic and hierarchical approach, which led to smaller networks and allowed for a more active optimization approach.

This issue is related to the very practical problem of how to actually manage a large project with thousands of activities. In reality, there are always interactions between activities; for instance, they may share some common resources (although, when possible, each resource unit should be dedicated to an activity until it completes). A prime example of a shared resource that cannot be fully dedicated is managerial attention. That is, activities must be coordinated and that requires someone to do the coordinating.* Most humans can coordinate between five to nine interacting activities. On the bright side, it is not necessary to coordinate all activities at once; complete activities and activities that are not due to start soon can usually be ignored. Nonetheless, in a project with thousands of activities, dozens of activities may be occurring in parallel, and some activities that have not been started still require managerial attention in advance. In particular, it is often necessary to stage resources for activities before they can actually start (as in Example 18.5), but some network models omit such staging activities, treating them as implicit. The point here is that these staging activities should take place while predecessor activities are still active, thus adding to the total managerial burden. As a result, it is impossible for a single manager to coordinate a large project well unless some tasks are delegated. The trick is to use the network structure and the PERT/CPM analysis to allocate the management of activities in a sound hierarchical manner. Clearly, if we can identify meaningful subprojects and replace them by single activities then such activities are really projects (albeit smaller than the parent project) which may require dedicated project managers (with or without the title). If we organize the parent project so that at any one time no more than about seven activities (or subprojects) run in parallel, then the project manager should be capable of coordinating the efforts of the individual subproject managers. Using the same principle, each subproject may have to be represented by sub-subprojects, and so on. This type of hierarchical aggregation constitutes one way to manage projects hierarchically, but it is not the only way. Another way would be for the project manager to focus on the most critical activities, using criticality metrics, and delegate the other activities. Each delegated activity then has to be managed to achieve a prescribed service level, as we discuss in Chapter 19.

PERT 21: A PERT/CPM Framework Fit for Use in the 21st Century

To be useful and usable, project scheduling and control framework should:

- (1) Yield reliable and relevant results
- (2) Be intuitively acceptable to decision makers, and
- (3) Require modest information inputs from users and provide easy-to-utilize outputs.[†]

* We discuss this issue in terms of human management, but we note that even computerized systems cannot coordinate too many entities without some kind of a hierarchy or networked processing.

[†] This implies that the input/output interface should be simple to use. However, we are not requiring the scheduling models themselves to be simple, as long as they are tractable. Such models should be

In spite of the spectacular historic success of PERT and CPM, for reasons we discussed extensively in the chapter and above, they do not fulfill the first condition. CPM simply ignores the stochastic nature of activity times; PERT accounts for them incorrectly and lacks the optimization approach of CPM. Perhaps for that reason, there has never been a serious attempt to use optimization methods alongside the PERT stochastic analysis. Furthermore, to the extent that providing triplet estimates is not only unreliable (Tversky and Kahneman, 1974) but also onerous (Woolsey, 1992), PERT fails to fully satisfy the third condition.

PERT 21 aims to satisfy all three conditions better than any existing framework. Again, the single most important difference between PERT 21 and traditional PERT/CPM is a stochastic analysis "engine" that is much more effective and also more efficient than the traditional one. The new engine provides reliable activity time distributions suitable not only for control but also for planning. These distributions make it practicable to address issues such as sequencing, scheduling due dates and release dates (that is, safe scheduling), and crashing by relevant stochastic models. That opens the door to a true merger of the historical contributions of PERT and CPM while avoiding the historical pitfalls. It is well known that PERT underestimates the project duration (by ignoring the Jensen gap). That problem is not an issue, however, when simulation is used. But the most pernicious flaw is the statistical independence assumption at the heart of PERT. The technical details of this point are addressed in a working paper available at this same website (Trietsch et al., 2010).

The paper analyzes field data from two Armenian project organizations as a basis for validating the new engine of PERT 21. One organization provided few projects but the data were sufficient to show that the lognormal assumption with linear association is at least as good as the conventional alternative. The insights gleaned from that analysis were then used to analyze a family of nine projects provided by the second organization. That case also involved the Parkinson effect, which makes the analysis slightly more complex and essentially requires simulation. However, the new engine does not aim to replace the need for simulation, but rather to provide more realistic simulation inputs. The final result indicates that PERT would grossly underestimate the true variance of the makespan, whereas the new engine leads to excellent results.*

Given this finding, one might ask why PERT was so successful historically. The answer is that PERT may be a poor *planning* tool but, apparently, it proved itself as a useful *control* tool, as suggested by the following quote: "the general observation can be made that in the long run the value of any control system has always depended on the individuals

computerized and, unless they take too long to converge, their internal complexity does not matter to the user.

* As is the case with any empirical study, the results should not be interpreted as "proof" that the model is correct. The study did show that the PERT assumptions did not hold. In a practical sense, that means that the new model is probably a better alternative. Further research, preferably by others, is required to validate, reject or improve the model. In this connection, we note that the research essentially studied ratios between actual activity durations (sorted by projects) and the original estimations of the same activities. Surprisingly, there is very little information in the literature about such ratios. Hill, Thomas and Allan (2000) did study the relationship between realizations and estimates in a software organization. The structure of their regression analysis was not sufficiently similar to that of Trietsch et al. (2010) to allow a direct comparison or to make possible testing the new model on their data. Their observations are compatible with the new model, but cannot be used directly to validate it.

who use it in making decisions[.] If they have confidence in the system, and feel that it aids them in the decision process, then there is strong and sufficient testimony as to its worth" (Malcolm et al., 1959, p. 668). A good control tool should help managers manage critical activities on an ongoing basis, and PERT evidently does that well. However, as part of planning, PERT is also used for long-term scheduling, starting with a schedule that serves as the initial benchmark for the subsequent control. Reliable scheduling and good control are not conflicting goals: more reliable schedules provide better benchmarks for subsequent control. Thus, a good schedule can increase the likelihood that the project will complete within the time and the budget originally allotted to it. Furthermore, good planning is the key to reliable estimates: during the control phase it is simply too late. Although we focus here on scheduling, similar principles apply to budgeting.*

PERT 21: Recommended Ingredients

Trietsch (2005) discusses the challenge exposed by the debut of a framework known as *Critical Chain Project Management*, popularized by Goldratt (1997).[†] The paper highlights the shortcomings of CCPM and provides a road map for the development of a system such as PERT 21. One recommendation is to consider stochastic variation during crashing and sequencing decisions. Another recommendation is to perform sequencing and crashing iteratively, starting with sequencing: until we sequence, we don't know which activities have a sufficiently large criticality to justify crashing. The paper also discusses the need to use statistical control principles in controlling projects. Such recommendations cannot really be implemented unless we have reliable activity processing time distributions, however, which is where the new stochastic engine comes in. PERT 21 advocates replacing the PERT stochastic engine without completely abandoning PERT/CPM. Except for that engine, PERT/CPM is a solid and tested framework. Furthermore, industry is heavily invested in PERT/CPM software, and that investment can and should remain productive. On the one hand, the new engine requires historical data of the type collected by PERT/CPM software, so it would fit best in existing information systems. On the other hand, it requires less direct input from the user. Thus it is also more efficient. But full-fledged PERT 21 implementations can go further than just replacing the old engine. The new engine also opens prospects for improved stochastic sequencing, scheduling and crashing models, thus finally making it possible to integrate the two parts of PERT/CPM. Furthermore, sequencing models that rely on neighborhood searches have recently been proposed for projects (Fleszar and Hindi, 2004), as we discuss in Chapter 17. Such models can be applied directly using stored samples and thus achieve true stochastic sequencing models. In this connection, the current version of the new engine generates activity time distributions that are lognormal and share the same coefficient of variation. Importantly, such variables are stochastically ordered: an activity with a higher estimated duration is stochastically larger. Stochastic sequencing models are often more tractable if they can rely on stochastic ordering (see Chapter 7). Thus, such models are supported by

* During the last decade, one of us followed the progress of two new academic buildings in two academic institutions. They exceeded their budgets by about 75% and 150% and were tardy by 30% and 80%, respectively. In both cases, the decision makers had no clue about the true range of possible outcomes. (Remarkably, only the one who missed by *less* maintained his dignity by resigning his position.)

[†] According to Ash and Pittman (2008), the original CCPM idea is due to Pittman (1994). We discuss CCPM in more detail in our Research Notes for Chapter 18.

the systemic error model. PERT 21 can also incorporate new graphical representations designed specifically for stochastic analysis and control (such as predictive Gantt charts, as we show in Chapter 19). All in all, PERT 21 has the potential to be the first truly effective stochastic planning and control framework for projects.

Several existing and prospective models become practicable once we obtain realistic distributions using the new stochastic analysis. In Chapter 19 we discuss how to utilize such distributions to obtain sample-based optimal results of various sorts. In our case, these realizations must be correlated as dictated by the systemic error model, but there is no conceptual difficulty in building such stored samples. Given a stored sample that represents reality sufficiently well, if we optimize various measures with respect to the sample, the result is approximately optimal in reality. There are two simple keys to the validity of the approach: (i) the sample must be based on realistic distributions; and (ii) the sample must be sufficiently large. Given modern computing power, it is easy to satisfy the second key. As for the first, our new engine is designed precisely to provide such realistic distributions. In Chapter 19 we discuss a safe scheduling model that optimizes release dates for activities that require staging inputs that should not be made available much earlier than might be necessary. That model indirectly sets optimal safety time buffers for the project by balancing criticalities. We also provide a model for stochastic crashing, and we sketch a model for booking expensive resources. The latter is essentially based on a hierarchical application of the release date optimization model. Such models should also become part of PERT 21. In addition, there is room to develop similar models for multiple projects.

An important future application of the new engine worth developing is the control of an ongoing project. As noted above, the initial success of PERT was predicated on its perceived success in aiding managers control ongoing projects. Similar short-term control decisions can be supported by the new engine without any adaptation. However, as a project progresses, we may evaluate the estimation bias that applies to it and apply further corrective actions beyond short-term control. We present a model for this purpose in our Research Notes for Chapter 19.

References

- Ash, R.C. and P.H. Pittman (2008) "Towards Holistic Project Scheduling Using Critical Chain Methodology Enhanced with PERT Buffering," *International Journal of Project Organization and Management* 1, 185-203.
- Baker, K.R. (1974) *Introduction to Sequencing and Scheduling*, Wiley, Hoboken, NJ.
- Baker, K.R. (2005) *Elements of Sequencing and Scheduling*, Tuck School of Business, Hanover, NH.
- Clark, C.E. (1961), "The Greatest of a Finite Set of Random Variables," *Operations Research* 9, 145-162.
- Clark, C.E. (1962), "The PERT Model for the Distribution of an Activity Time," *Operations Research* 10, 405-406.

- Demeulemeester, E. and W. Herroelen (2002) *Project Scheduling: A Research Handbook*, Kluwer Academic Publishers.
- Dodin, B. (1985) "Bounding the Project Completion Time Distribution in PERT Networks," *Operations Research* 33, 862-881.
- Elmaghraby, S.E. (1977) *Activity Networks: Project Planning and Control by Network Models*, Wiley.
- Ford, L.R. and D.R. Fulkerson (1962) *Flows in Networks*, Princeton University Press.
- Fulkerson, D.R. (1961) "A Network Flow Computation for Project Cost Curves," *Management Science* 7, 167-178.
- Fulkerson, D.R. (1962) "Expected Critical Path Lengths in PERT Networks," *Management Science* 8, 808-817.
- Goldratt, E.Y. (1997) *The Critical Chain*, The North River Press, Great Barrington, MA.
- Grubbs, Frank E. (1962) "Attempts to Validate Certain PERT Statistics or Picking on PERT," *Operations Research* 10, 912-915.
- Herroelen, W., R. Leus and E. Demeulemeester (2002) "Critical Chain Project Scheduling: Do Not Oversimplify," *Project Management Journal* 33(4), 48-60.
- Hill, J., L.C. Thomas and D.E. Allen (2000) "Experts' Estimates of Task Durations in Software Development Projects," *International Journal of Project Management* 18, 13-21.
- Kelley, J.E. Jr. (1957) "Computers and Operations Research in Road Building," *Operations Research, Computers and Management Decisions Symposium Proceedings*, Case Institute of Technology, Jan 31-Feb 1, 2.
- Kelley, J.E. (1961) "Critical-Path Planning and Scheduling: Mathematical Basis," *Operations Research* 9, 296-320.
- Kelley, J.E. (1963) "The Critical Path Method: Resources Planning and Scheduling," Chapter 21 in J. Muth and G.L. Thompson *Industrial Scheduling*, Prentice-Hall, 347-365.
- Kelley, J.E. Jr. and M.R. Walker (1959) "Critical Path Planning and Scheduling," *Proceedings Eastern Joint Computer Conference*, Boston, December 1-3, 160-173.
- Klingel, A.R. (1966) "Bias in PERT Project Completion Time Calculations for a Real Network," *Management Science* (Series B) 13, B-194-201.

- Leach, L. (2003) Schedule and Cost Buffer Sizing: How to Account for the Bias between Project Performance and Your Model, *Project Management Journal* 34(2), 34-47.
- MacCrimmon, K.R. and C.A. Ryavec (1964) "An Analytical Study of the PERT Assumptions," *Operations Research* 12, 16-37.
- Malcolm, D.G., J.H. Roseboom, C.E. Clark and W. Fazar (1959), "Application of a Technique for a Research and Development Program Evaluation," *Operations Research* 7, 646-669.
- Pittman, P.H. (1994) "Project Management: A More Effective Methodology for the Planning & Control of Projects," Doctoral dissertation, University of Georgia.
- Premachandra, I.M. (2001) "An Approximation of the Activity Duration Distribution in PERT," *Computers & Operations Research* 28, 443-452.
- Sasieni, M.W. (1986) "A Note on PERT Times," *Management Science* 32, 1652-1653.
- Schonberger, R. (1981) "Why Projects are 'Always' Late: A Rationale Based on Manual Simulation of a PERT/CPM Network," *Interfaces* 11(5), 66-70.
- Trietsch, D. (2005) "Why a Critical Path by any Other Name would Smell Less Sweet: Towards a Holistic Approach to PERT/CPM," *Project Management Journal* 36(1), 27-36.
- Trietsch, D. (2005a) "The Effect of Systemic Errors on Optimal Project Buffers," *International Journal of Project Management* 23, 267-274.
- Trietsch, D., L. Mazmanyan, L. Gevorgyan and K.R. Baker (2010), "A New Stochastic Engine for PERT," Working Paper.
- Tversky, A and D. Kahneman (1974) "Judgment Under Uncertainty: Heuristics and Biases," *Science* 185, 1124-1131.
- Van Slyke, R.M. (1963) "Monte Carlo Methods and the PERT Problem," *Operations Research* 11, 839-860.
- Wiest, J.D. (1964) "Some Properties of Schedules for Large Projects with Limited Resources," *Operations Research* 12, 395-418.
- Wiest, J.D. (1967) "A Heuristic Model for Scheduling Large Projects with Limited Resources," *Management Science*, 13, B359-377.
- Woolsey, R.E. (1992) "The Fifth Column: The PERT that Never Was or Data Collection as an Optimizer," *Interfaces* 22(3), 112-114.

Zhao, T. and C.-L. Tseng (2003) "A Note on Activity Floats in Activity-On-Arrow Networks," *Journal of the Operational Research Society* 54, 1296-1299.