Research Notes for Chapter 19[*]

Chapter 19 is based on the first edition's Chapter 18, but Section 19.3—on hierarchical balancing and progress payments—is essentially new in the second edition: only the coverage of Example 19.5 had appeared previously. The first part of the material below accompanied Chapter 18 of the first edition, repeated here with very minor edits. The second part extends the coverage of the chapter by presenting additional balancing models. It is largely motivated by recent results obtained for an alternative project balancing framework that is sometimes more appropriate (especially for small projects).

In the first part, following sources and comments for the results in Chapter 19, we provide the proof of Theorem 19.1 and its extension to discrete distributions. Next, in practice, project schedules have to be monitored and controlled during execution. In particular, even if our initial schedule is stochastically balanced, the schedule of the remainder of the project may require corrective actions to stay in balance. One way to do that is to update release dates and crashing decisions based on feedback. But we should not forget that processing times are not likely to be statistically independent; that is, at any particular stage, our processing time estimates for the remainder of the project can benefit from information revealed in the previous stages. We discuss this issue briefly and present a new graphical tool for focusing on the most important control actions. Control is a rich subject, however, and it justifies further research, including validation of the model we present. Then, we describe and critique Critical Chain Project Management (CCPM), which we briefly introduced in our Research Notes for Chapter 16. To do so, we must also discuss the "Theory of Constraints" (TOC), because CCPM relies on it. In fact, the TOC principles are inherent in conventional PERT/CPM, so this reliance demonstrates that the essence of CCPM is not new. But it is sufficiently different from conventional PERT/CPM to justify specific attention. The main merit of CCPM is that its commercial success highlighted the practical need to enhance the conventional PERT/CPM framework, but its pitfalls are rooted in serious theoretical errors. We also compare CCPM to PERT 21, which both rely on PERT/CPM and promote a simple user interface. But they are distinct with respect to the use of historical data (missing in CCPM), buffer setting, crashing (also missing in CCPM), hierarchical approach (again missing in CCPM), and control.

The extensions covered in the second part allow delaying expenditures of delayed activities, whereas the chapter's model assumes expenditures start at their activity release dates even if the activities are delayed. Although such models have been studied since the mid-1980s, our coverage here is motivated by new results that have been published for them after our text went to print. One contribution here is that we cast these results in terms of sample-based analysis, But we also provide a practical approach to the solution of less restricted models,

[*] The Research Notes series (copyright © 2010, 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 (2024) Research Notes for Chapter 19 in *Principles of Sequencing and Scheduling*, 2nd ed. (Wiley, 2019).
URL:http://faculty.tuck.dartmouth.edu/principles-sequencing-scheduling-2e/notes/.

including ones that do not require statistical independence and a mixed model that distinguishes explicitly between activities that require preparation (and thus incur expenses at their release dates) and ones that do not incur costs until started.

In our conclusion, which follows the second part but had appeared already in the notes of the first edition, we discuss an independent NASA case study where some of the stochastic models that we incorporate in PERT 21 were successfully implemented.

Sources and Comments

Because Chapter 19 is focused on safe scheduling, our Research Notes for Chapter 7 are also relevant here and should be consulted together with this installment. For completeness, however, we repeat some details that are most directly relevant to project scheduling. In a sense, the earliest safe scheduling model harks back to Malcolm et al. (1959), who showed how to calculate service levels. Adjusting the due date to achieve a desired service level is then easy. As we discussed in Chapter 16, however, their model is theoretically flawed, mainly because it ignores the Jensen gap and assumes statistical independence. On the one hand, because we treat the simulation approach of Van Slyke (1963) as part of PERT, the Jensen gap can be addressed easily.[*] On the other hand, the quality of simulation results depends on the quality of the information behind it; that is, unless processing time distributions are valid, simulation cannot estimate the Jensen gap correctly. Nor can we rely on due dates set by such simulation to provide the required service level. The same observation applies to the use of approximations to estimate the Jensen gap, such as the one introduced by Clark (1961). Hence, we must also use legitimate estimation for processing time distributions, and account for correlations.

Considering the economic approach to safe scheduling, Britney (1976) adapts the critical fractile model for project activities. However, he essentially allocates to each activity its own safety time. In effect, he uses a single-operation approach. The working assumption is that if an activity does not complete on time, it causes problems downstream that can be adequately modeled by the activity's individual tardiness cost. By contrast, if an activity is early, the activities that follow it wait for the originally scheduled completion date, which acts as their release date (unless another predecessor has a later scheduled completion date). Thus, each activity acquires a Parkinson processing time distribution (Appendix A). This makes it possible to use the basic newsvendor model for each activity individually. In our Research Notes for Chapter 7 we noted that this model is even simpler than the single-machine *n*-job model. As such, it is certainly simpler than the project stochastic balance (PSB) model that we introduced in this chapter. Britney also discusses stochastic crashing, under the assumption that crashing changes the mean of the distribution but not its variance (or its shape). Another stochastic crashing model that relies on the same assumption is given by Wollmer (1985). Wollmer seeks to minimize a positively weighted sum of the expected duration and the crashing cost. To enable solution by integer programming, he also assumes that distributions are discrete. On the one

[*] PERT's development was not supposed to stop with the publication of Malcolm et al (1959). Accordingly, we consider simulation and published approximations for the Jensen gap as legitimate parts of PERT. Likewise, the ACM model was originally developed to support PERT. In this vein, once we add explicit CPM-like modules for sequencing and for crashing to PERT, we can also treat them as part of PERT. Equivalently, we sometimes use the composite term PERT/CPM to refer to the union of both approaches. Some authors, however, prefer strict distinction, and they refer to CPM, PERT and simulation as three distinct approaches. (Some of these authors also claim that the AON approach is part of CPM whereas AOA is associated with PERT. Historically, however, both PERT and CPM originally used AOA [Malcolm et al., 1959; Kelley, 1961]—probably because it lent itself to more convenient computer programming—and both can be implemented with AON networks.)

hand, Wollmer's approach is more holistic than Britney's because he accounts for the network structure explicitly. On the other hand, the approach is computationally complex. Models that extend the newsvendor model to *n* jobs without ignoring the interactions between them did not emerge until the mid-1980s. One stream of research concerns *n* activities in series (in a supply chain or project context). We may refer to each of the *n* activities as *stages*. As we noted in our Research Notes for Chapter 7, that model is essentially equivalent to processing *n* jobs on a single machine with a makespan objective. The assembly coordination model (ACM) model, which we presented in the chapter, was introduced independently several times, including Ronen and Trietsch (1988), Kumar (1989), and Chu et al. (1993). Specifically, Ronen and Trietsch use a project context and address the constrained case explicitly, whereas the others use a supply chain context without such constraints. Trietsch and Quiroga (2004) compile these results and provide new bounds. Hopp and Spearman (1993) introduce an alternative version of the ACM, with a step cost function for tardiness. Trietsch (1993) considers both types of tardiness cost function in the context of scheduling a hub airport (as briefly described in Chapter 1). His model can also be applied to a very basic multiple project environment, because it coordinates multiple related flights each with its own earliness and tardiness penalties. Incidentally, the cost structure of Example 19.2 is similar to that used by Trietsch (1993) and provides a practical example of a case where different jobs require different service levels.

As we discussed in our Research Notes for Chapter 7, models for safety time, of the type we seek in safe scheduling applications, have roots in inventory theory. Specifically, we showed that the critical fractile rule can be presented as a special case of a stochastic balance inventory rule developed by Arrow, Harris and Marschak (1951). There are other applications of stochastic balance in addition to safety stock and safety time models. Specifically stochastic balance models also apply for capacity-setting and for financial engineering. A straightforward financial application is quoting a fixed price for a project. A more complex model arises when the quote is part of an auction; that is, when several contractors submit quotes—or bid—on the same project and the lowest bidder wins. Every bidder estimates the cost and adds a profit margin, but the cost estimates are subject to stochastic error. If the quote is too low to make a profit, it is likely to win (so a loss is incurred), whereas a quote that is excessive could make a tidy profit but is not likely to win the contract. In such circumstances, we need safety not only for stochastic cost overruns but also for stochastic estimation errors. To our knowledge, however, this model has not been researched yet. Revenue management models, where the price of perishable goods—e.g., airline seats for a particular flight or hotel rooms for a particular date—is adjusted dynamically to maximize profit, also rely on stochastic balance. The capacity setting model presented by Trietsch and Quiroga (2009) is the dual of the ACM. Specifically, capacity is dual to processing time, in the sense that by increasing capacity we can achieve crashing. Trietsch and Quiroga (2009) also elaborate on the case where a distribution is transformed by capacity adjustments beyond just changing its mean. Equation (19.9) is based on their template. They also provide an effective heuristic for this purpose that is robust for any λ between 0 and 1. Trietsch (2012) analyzes the case of crashing a serial project. Finally, the PSB model is due to Trietsch (2006).

In Chapter 9 we discussed balancing load on parallel machines and observed that to minimize the expected weighted sum of the due date and tardiness (which implies stochastic balance), we must not balance the expected load on the machines but rather balance their criticalities. We did not consider precedence relations, however. Suppose we are asked to partition a set of tasks with precedence relationships among them (as in a project) to *m* subsets such that the tasks of subset *k* have no predecessors in subsequent subsets ($k + 1, ..., m$). In other words, if all subsets (1, 2, ..., $k - 1$) have been completed, subset k is feasible (but we must satisfy all precedence relationships between pairs of jobs *within* the subset). Therefore, it is possible to allocate the tasks to m stations in a line, and the cycle time is then determined by the most loaded station. This model generalizes the parallel machine model by adding precedence constraints. It is also a version of the *line-balancing* model. In the common formulation of the line-balancing model the parameter m is a decision variable; the objective is minimizing m subject to a constraint on cycle time. However, the ability to solve the parallel machine model with precedence constraints for any m is sufficient to solve the common formulation, too. Whereas the deterministic line-balancing model has been studied extensively, the safe scheduling versions still require research.

Theorem 19.1: A Proof

In the chapter we posed Theorem 19.1 under the assumption that activity time distributions are continuous. However, we also showed how to extend the theorem to discrete distributions. Here we prove both the continuous version and the discrete one. Starting with the former, we also require processing times not to be perfectly correlated (as they might be if they were linearly associated with the common factor as the only proper random element).[*] When some processing times are perfectly correlated it is possible that two or more of them become critical, or not, together. The formal optimality conditions in such a case are obtained as in the discrete case, so we include that case there. In both cases we also require the distributions to be static; that is, processing time must not be functions of start times or sequence positions. For continuous distributions, if we insert the conditions into the theorem itself, we obtain the following version:

Theorem RN19.1: When activity times are stationary, continuous, and none of them are perfectly correlated, the following are necessary and sufficient optimality conditions for the PSB problem with probability 1.

1. $q_j^* = v_j^*/(\alpha + \beta) \geq \alpha_j/(\alpha + \beta)$; $j = 1, 2, \ldots, n$
2. if $v_j^* > \alpha_j$ then $r_j^* = ES_j$
3. $q_{n+1}^* = 1 - \sum_{j=1}^{n} v_j^*/(\alpha + \beta) \leq \alpha_{n+1}/(\alpha + \beta)$.

where v_j* reflects the true marginal economic implication of postponing r_j per time unit.

Proof.

»» The proof has two parts. In the first part we show that the theorem implies the Karush-Kuhn-Tucker (KKT) conditions. The KKT conditions are necessary for a local minimum. They become sufficient when the model is convex (because any local optimum must then be global), and that's what we show in the second part.

[*] Trietsch (2006) allows perfect correlation but his definition of q_j^* is slightly different.

To demonstrate that the theorem conditions are indeed KKT, observe that the probability that two release dates are critical at the same time is zero. That is, the critical release date is unique with probability 1. Consider the case where $r_j^* > ES_j$ and thus $v_j^* = \alpha_j$. Here, we require $q_j = \alpha_j/(\alpha + \beta)$, which we prove first. For any realization of the stochastic processing times and any set of release dates, r_j is either critical—with probability q_j—or, by the assumption, there exists a positive but sufficiently small ε by which we can postpone r_j without rendering it critical. Our expected gain by such a postponement is $(1- q_j)\alpha_j\varepsilon$ and our expected loss is $q_j(\alpha + \beta - \alpha_j)\varepsilon$. Setting $q_j^* = \alpha_j/(\alpha + \beta)$ balances the expected gain against the expected loss; that is, the gradient of the objective function becomes zero, which is one way to satisfy the KKT conditions. Next, consider the case where some $r_j^* = ES_j$ due to constraint. Postponing a release date can only increase its criticality, so that can happen only if the criticality exceeds $\alpha_j/(\alpha + \beta)$. Therefore, in this case, $v_j^* > \alpha_j$, which is the other way to satisfy the KKT conditions. Thus, subject to our assumption, we proved (1) and (2). (3) follows directly to satisfy $\Sigma_{j=1,\dots,n+1}q_j^* = 1$. This completes the demonstration that the conditions are equivalent to KKT.

To prove convexity, recall our mathematical program (the PSB),

$$\text{Minimize } Z = E\left[\sum_{j=1}^{n+1}\alpha_j\left(C - r_j\right)\right] = (\alpha+\beta)E\left(C\right) - \sum_{j=1}^{n+1}\alpha_j r_j \qquad (19.2)$$

Subject to

$$C_j \ge r_j + p_j; j = 1,\dots,n \qquad (19.3)$$

$$C_j \ge C_k + p_j; \forall k \in P(j), j = 1,\dots,n \quad (19.4)$$

$$C \ge d = C_{n+1} \quad (19.5)$$

$$C \ge C_k; \forall k \in P(N) \qquad (19.6)$$

Except for the constraint $C \ge d$ (19.5)—which, being convex, cannot void convexity—in this form the program is identical to the stochastic crashing model studied by Wollmer (1985) where the objective was minimizing the (positively weighted) expected completion time plus crashing costs. In (19.2) the expected completion time is weighted by $(\alpha + \beta)$ and $-\sum\alpha_j r_j$ can be interpreted as the crashing cost (because, as Figure 19.2[*] demonstrates, release dates are mathematically equivalent to any other project activity, so releasing an activity later constitutes negative crashing). Wollmer has shown convexity by using a general result by Wets (1966) that requires statistical independence between release dates and processing times but not among processing times. ««

When activity times are discrete, two or more release dates may be co-critical. Following the convention introduced in the chapter, we interpret q_j as the probability r_j is critical, regardless of whether other release dates are also critical. For instance, in the optimal solution of Example 19.2, four out of ten scenarios involve multiple critical release dates (see Table 19.4 and recall that the due date is also counted as a release date). The same may apply when some pairs of activities are perfectly correlated (with a coefficient of +1) and, again, in such a case all co-critical activities are counted as critical. Criticality is a function of the full set of release dates, but we write $q_j(r_j)$ to denote it as a function of its own release date when all other release dates

[*] Reproduced as Figure RN19.2 in the second part of these notes.

are held constant. Discrete distributions with more than one possible realization always have positive increments between successive values (typically, the increments are of one unit). In what follows, if ε is smaller than the smallest increment of any processing time distribution, we say that it is *sufficiently small*. For the case of continuous distributions that may be perfectly correlated, ε is considered sufficiently small for a realization if it is smaller than the amount by which the set of critical activities (there may be more than one activity in that set) has to be advanced to an earlier time before another set of activities becomes critical. Theorem 19.2 is essentially an elaboration on the conditions given in the chapter, namely that an optimal release date is the smallest feasible value for which $q_j > \alpha_j/(\alpha + \beta)$. These conditions are really a generalization of Theorem 19.1 in the sense that they also apply to continuous distributions. (The reason we chose to present the more restricted version is that it highlights the essence of stochastic balance.)

Theorem RN19.2: When activity times are stationary, discrete or continuous, and ε is sufficiently small, the following are necessary and sufficient optimality conditions for the PSB problem with probability 1 (for $j = 1, 2, \ldots, n$).

1. $q_j^*(r_j^*) \geq \alpha_j/(\alpha + \beta)$
2. If $r_j^* > ES_j$, then $q_j^*(r_j^* - \varepsilon) \leq \alpha_j/(\alpha + \beta)$

Proof.

»» As in the proof of Theorem RN19.1, we need to show that the conditions imply a local optimum and that the problem is convex, thus rendering the local optimum global. Convexity is assured by the same argument given for Theorem RN19.1, so we just have to show that the conditions imply local optimality. To demonstrate that the conditions imply a local optimum, we start with (1) and for contradiction, we assume $q_j^*(r_j^*) < \alpha_j/(\alpha + \beta)$. If we increase r_j by ε we save $\alpha_j\varepsilon$ with a probability of $(1 - \alpha_j/(\alpha + \beta))$ at least but lose $(\alpha + \beta - \alpha_j)\varepsilon$ with a probability of $\alpha_j/(\alpha + \beta)$ at most. The expected net gain is thus nonnegative. This completes the proof of (1). To prove (2), observe that $q_j^*(r_j^* - \varepsilon)$ can only be smaller than $q_j^*(r_j^*)$ if $r_j^* > ES_j$. If so, similar analysis can now be used to show by contradiction that if the condition does not hold we should increase r_j^*. (If the distribution is discrete, (1) can also be set as a strict inequality, which is what we did in the text.) ««

Schedule Control

Actual project performance is not likely to follow the initial schedule or budget precisely. Here, we limit ourselves to schedule control, subject to the assumption that our budget has a buffer that allows us to take reasonable corrective actions to control the schedule. The same principles can also be used for budget control. In fact, the two are intertwined, because schedule control affects budget consumption, but we don't know of an appropriate unified model.[*] In

[*] Song, Yano & Lerssisuriya (2000) study the correct buffering of two related factors, namely time and quantity. The model can be adapted for time and cost, however. They show that the model is not convex but a good approximation is obtained by treating the two separately. They do not discuss control, however.

general, we encounter two types of variation. The first is due to randomness of processing times. We prepare for this type of variation in typical safe scheduling models, and we depict it with a predictive Gantt chart. If only that type of variation occurs, we may say, loosely, that the project is under statistical control (which means that the variation is within statistically predictable limits). Another type of variation is due to changes in plans or other "out of control" events. That second type of variation typically requires re-sequencing as well as re-scheduling of release dates. By contrast, when the project is in statistical control, the safety time buffers are designed to reduce the need to re-schedule and drastically reduce the need to re-sequence.

Our historical regression model is described in Trietsch et al. (2010); see also our Research Notes for Appendix A. In that model, and below, we assume the use of lognormal approximations for the sum of all estimates of complete activities, denoted X, the sum of realizations, denoted Y, and the common factor, Q. We assume that processing times are linearly associated (see Appendix A), so $Y = QX$. Because X is a sum of independent random variables, we also employ the lognormal sum approximation (Appendix A). In more detail, for a given set of complete activities, X is modeled by a unique lognormal distribution whose mean is the sum of the estimated means of the activities and whose variance is obtained by adding up the variances of each activity, calculated by the estimated common coefficient of variation in our model, multiplied by the estimates and squared. The distribution of the common factor element is modeled as lognormal, too, and its parameters are estimated directly from our historical regression analysis. Because both distributions are lognormal, Y is also lognormal, and it is convenient to use a logarithmic scale. We denote the (normal) density function of the logarithm of the estimate by $f_X(x)$, where x represents the appropriate logarithm. We denote the (normal) density function of the common factor element by $f_Q(q)$, where q is also a logarithm. If the total time required for the set of completed activities is exp(y) (so its logarithm is y), then $x + q = y$; that is, $q = y - x$. The density of the event that log $Q = q$ given that log $Y = y$ is therefore proportional to $f_X(x)f_Q(y - x)$. To scale this expression and transform it to a cdf, we write:

$$F_{Q|Y}(q \mid \log Y = y) = \Pr\{\log Q \le q \mid \log Y = y\} = \frac{\int_{y-q}^{\infty} f_X(x) f_Q(y-x)dx}{\int_{-\infty}^{\infty} f_X(x) f_Q(y-x)dx}$$

This conditional distribution can be used for estimating distributions for the remainder of the project. As we progress, the distribution of X becomes tighter (smaller *cv*), and therefore the distribution of Q also becomes tighter. We note, however, that all estimates are based on the resource allocations (or modes) originally planned, so if, as part of control, we allocate more or less resources to an activity, its original estimate has to be corrected to avoid confounding the effect of Q with the effect of capacity allocation.

Using such calculations, we can update distributions at any stage during the project execution. We use them to update plans and for control decisions. As a rule, we may expect our predictive Gantt charts and our associated *flag diagrams*—which we introduce presently—to change dynamically as distributions are updated. The question is what to do with the information. Although a fully objective answer may not exist, in our opinion, we should not re-sequence after every event that causes the predictive Gantt chart and the flag diagram to change. However, because scheduled release dates may entail preparation that is not depicted by the project network, our flexibility to change plans during the execution (control) phase may be limited. When necessary, therefore, we may have to re-plan the remainder of the project, including re-sequencing. But if we still have flexibility, our approach is to continue adjusting the system—to achieve projected economic balance into the future. That is, the usual response should be restricted to rescheduling of future release dates where no investment has been made yet. (The mixed model we introduce in the second part of these notes is inherently more flexible with respect to activities that can be started without major preparations.)

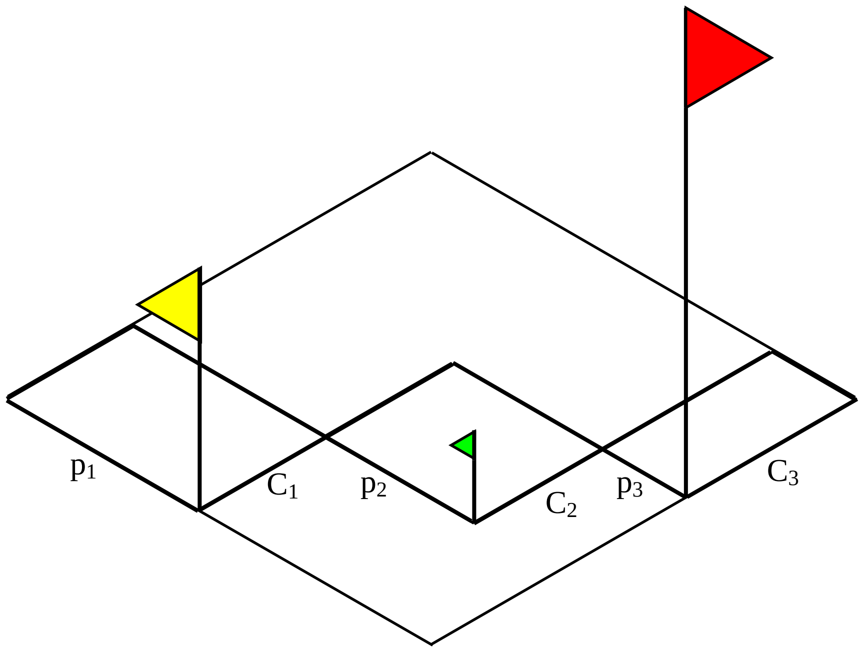

Figure RN19.1: *A Flag Diagram*

A good decision support system (DSS) must (i) provide a clear signal when a correction is required, (ii) facilitate the interactive determination of an approximately correct response, (iii) provide a check on the new plan, and (iv) update the plan once the proposed change is accepted by the user. A new plan is often required when the project scope is changed, or when it becomes clear that the original plan is no longer viable; for instance, if an important development fails and we must use a different approach. All these require further research, but the flag diagram—introduced by Trietsch (2003), and in more detail by Arthanari & Trietsch (2004)—is designed specifically to provide graphical signals showing the magnitude and orientation of deviations from balance that are not due to scope change etc. Thus, it supports these DSS requirements.

The flag diagram is based on the same distributional information as the predictive Gantt chart, but it provides signals. The predictive Gantt chart, in contrast, merely shows the projected probabilities of completion of various events. Figure RN19.1 illustrates. In this figure, for each activity, a rectangle (shown as a parallelogram) depicts the criticality and the normalized marginal cost. If the two are equal or close (that is, the rectangle is a square), the activity is in stochastic balance. Otherwise, flags whose height is proportional to the marginal opportunity in rebalancing are used to highlight which of the activities are most out of balance. In our context, the activities whose criticalities sum to one are future release dates and ongoing activities (including the due date, r_{n+1}). These activities are a subset of the current uniformly directed cut

(UDC).[*] The color scheme in the figure is determined somewhat arbitrarily as a function of the marginal gain available by providing balance; that is, tall flags are red, short ones are green. The size of a flag is also important because it provides partial evidence with respect to the size of the opportunity. The red flag in this particular example is associated with an activity whose criticality is too low (criticalities are denoted by p_j in this figure), which is why it is oriented to the left. In our context, this indicates that the release date of the activity can be postponed. If the activity has already started, however, it would be beneficial if this activity could loan resources to the one with the yellow flag, whose criticality is excessive. Alternatively, if the release date of the yellow flag is still in the future, we should accelerate it if possible. But it might be best to leave the green-flag activity alone: it is nearly balanced, and its orientation may even be reversed as a result of an adjustment elsewhere. Nonetheless, the best control action may be to change a downstream activity. The flag diagram can highlight problems and opportunities but it cannot by itself determine the best response.

There are two potential signals of trouble. First, it may become impossible to maintain ideal balance if any release date becomes constrained; that is, it becomes too critical even if scheduled at its ES. If so, the due-date performance of the project may be at more risk than originally anticipated. Here, the role of the DSS is to highlight the increasing risk and to help identify activities that present opportunities for remedial action (crashing). The activity that the due date in question controls may be one of them, but the best opportunity may be further downstream. The current version of the flag diagram cannot show such downstream opportunities, so developing it to be capable of doing so is an open research challenge. The second type of signal occurs when an activity is delayed beyond its release date so that the project due date performance is at excessive risk (again). If other activities are available for processing on the resources that are idle due to the delay, it may be beneficial to dispatch another activity ahead of its turn. Here the role of the DSS is to alert the user to the problem and to the opportunity. In this case it is necessary to estimate how long the delay is likely to continue (an estimate that we may treat like any other estimate—including bias correction and variance allocation).

Critical Chain Project Management and Management by Constraints

Critical Chain Project Management (CCPM) is generally attributed to Goldratt (1997), but according to Ash & Pittman (2008) the credit is due to an unpublished PhD dissertation, Pittman (1994). Judging by the number of enthusiastic papers about CCPM, one might think it is universally successful and meritorious, but in fact, there are several problems with it. Papers that expose such problems include Herroelen & Leus (2001), Herroelen, Leus & Demeulemeester (2002), Raz, Barnes & Dvir (2003), Trietsch (2005), and Millhiser & Szmerekovsky (2008).

To describe CCPM, we must start with the so-called Theory of Constraints (TOC)—e.g., see Goldratt (1990). This "theory" is actually nothing more than a useful meta-heuristic. There are many reports of successful applications although that evidence is anecdotal. However, it turns out that Goldratt interprets one of the steps of TOC in a way that can lead to serious suboptimality, by emphasizing that the binding constraint is unique. There is no evidence that the successful applications took Goldratt's interpretation literally. For this reason, we adopt the term *Management by Constraints* (MBC)—e.g., see Ronen and Starr (1990)—to refer to the useful

[*] As we indicated in our Research Notes for Chapter 16, the current UDC may also include some complete activities if their finish event also relies on other UDC activities. Such complete activities are also called *dormant*. So our set here comprises the current UDC minus any dormant activities in it.

interpretation of the steps in TOC (but we continue to use TOC to refer to Goldratt's version). MBC has six steps (although the first, denoted by 0, is rarely counted explicitly):

0. Select an objective function and decide how to measure it
1. Identify the binding constraint(s) that limit our ability to improve the objective function
3. Manage the binding constraint(s) to improve the objective function maximally[*]
4. Subjugate everything else to the needs of the binding constraint(s)
5. Elevate the binding constraint(s)
6. Return to Step 1, since the binding constraint(s) may have changed after Step 4.

Although the possibility of multiple binding constraints is acknowledged, the way the heuristic is promoted implicitly suggests the existence of just one binding constraint, or *bottleneck*. This interpretation is misleading in the sense that it oversimplifies the analysis.

In interpreting the MBC steps, one might surmise that applying step (4) repeatedly would eventually lead to some kind of balance between the main resources. And here lies the problem: TOC forbids balance and seeks to keep the nominal bottleneck busy 100% of the time regardless of cost (see Trietsch, 2003, 2005a). If we were to follow that prescription to the letter, investments in maintaining excess capacity on non-bottleneck resources would be, in the limit, completely wasted. Hence, to avoid such waste, MBC must not forbid resource balance. Indeed, nothing in the five steps prohibits balance. That explains why MBC works, but we should go one step further and seek the most economical balance; that is, we should seek stochastic economic balance. To that end, Trietsch (2005a) extends Management by Constraints to *Management by Criticalities* (MBC II). The main idea is that no single resource should be allowed to be a consistent active constraint. Instead, the correct criticality of each resource (that is, the frequency at which it limits the objective function) should be proportional to its economic value. Trietsch (2007) shows how to implement MBC II in hierarchical systems.

Whereas MBC is usually associated with capacity management, the same general approach works with a time focus. Indeed, MBC is isomorphic to the fundamental CPM approach. To demonstrate, we cast CPM as MBC steps:

0. [Select an objective function and decide how to measure it] The objective is to minimize the project duration, and it is measured by the makespan.
1. [Identify the binding constraints that limit our ability to improve the objective function] The constraints that limit the project duration are activities on the critical path (but recall that the critical path can include activities in series that could have been carried out in parallel except due to resource constraints)
2. [Manage the binding constraints to improve the objective function maximally] The activities on the critical path must be performed without any delay, and any earliness should be utilized
3. [Subjugate everything else to the needs of the binding constraints] All other activities must start before their latest start time, to support the critical path. Often, this is interpreted conservatively and all activities start at their earliest start time, thus subjugating their timing to the needs of the binding constraints

[*] We are not quoting Goldratt verbatim. In this case his words were "Decide how to exploit the constraints," but we assume he had not intended a restriction of this to planning only. Our version includes planning, execution, and control.

4. [Elevate the binding constraints] The most cost-effective crashing opportunity is pursued to improve the makespan (in practice, crashing often implies adding resources)
5. [Return to Step 1, since the binding constraints may have changed after Step 4] During crashing, the composition of the critical path can change, so before selecting another crashing opportunity we must re-identify the critical path, thus returning to Step 1.

In other words, the crashing process of CPM constitutes an MBC cycle. Thus PERT/CPM and MBC are isomorphic.

MBC is not the first application of this focusing method in the context of maximizing throughput either. For instance, Bock (1962) presents an early approach to teaching linear programming by focusing on binding constraints.* Trietsch (2005) and Trietsch (2005a) list additional evidence that the principles of MBC were well understood and practiced before it was ever articulated. Thus, when presented by Goldratt, the MBC idea was no longer revolutionary. Nonetheless, the articulation of MBC in simple English is a remarkable achievement (Balakrishnan, Cheng & Trietsch, 2008).

Given the isomorphism between PERT/CPM and MBC, there is no need to "apply" MBC to project management. Nonetheless, CCPM was developed with the clear intent of doing just that. As with TOC, the development of CCPM stresses simplicity at the expense of correctness. The justification is a claim that the results are sufficiently good and that opting for correct theory instead would not be worth the effort. However, that claim has never been validated; arguably, the opposite is often true. CCPM starts with a standard network presentation of the project. Usually, that network is depicted in the format of a Gantt chart, but precedence arrows are shown explicitly. Resource constraints have to be resolved before drawing the network diagram.

The next step in CCPM scheduling is to provide safety time buffers. The exact procedure is rarely, if ever, explained in sufficient detail, but the published examples are based on a hidden assumption that projects have an assembly tree structure. However, it is not particularly difficult to remove this assumption.

The departure point in CCPM—starting with Pittman (1994)—is an assumption that activities are typically estimated with an implicit safety time buffer added; usually, the assumption is that this buffer is 100% of the initial estimate (that is, the buffer is 50% of the final estimate). Again, this assumption has never been validated, and there is hard evidence that it is often wrong. Hill, Thomas & Allen (2000) cite a software company case where the mean bias of project estimates was below 1%. Their data show that underestimation errors tended to be larger than overestimation errors (indicating a distribution skewed to the right), so although overestimation was more prevalent, the final bias was very small.† In a nine-project case studied

* Bock uses a sample problem of optimizing a mix of products subject to constraints. Sorting products by profit per unit, he first identifies the maximal amount that can be produced of the most profitable product by the constraint that is most binding for it (the "bottleneck"). Then, additional products are introduced in order of profitability, to utilize any idle resources that remain. Lastly, tradeoffs are identified on bottleneck resources that increase the profit. Because the approach is not sufficiently powerful for cases that involve solving sets of two or more equations, Bock recommends it only as an initial step in teaching LP. TOC utilizes a streamlined version of Bock's approach where products are introduced by the order of their contribution per unit of the bottleneck, thus avoiding the need to use tradeoffs for corrections. The streamlined version is still not sufficient to address models with multiple bottlenecks (Balakrishnan & Cheng, 2000).

† That observation is commensurate with our own assumption of lognormal distributions. However, the authors did not collect data on ratios of realizations to estimates; instead they focus on differences. Therefore, their data cannot be used directly to validate our assumption. The paper also reports a significant difference between the average errors of an apparently optimistic manager and an apparently pessimistic one. That observation is commensurate

by Trietsch et al. (2010), the bias was strongly in the opposite direction to CCPM's assumption: estimates were consistently optimistic (no single activity completed on time). Thus, the CCPM assumption of highly padded estimates is not universally valid.

The proposed buffering approach in CCPM has two parts. First, each activity estimate is cut in half, to remove the alleged hidden safety time. Next, for the activities on the critical path, one quarter of their total initial estimate—or half the time now allowed—is appended as a *project buffer* at the end of the project. The due date is thus set at 75% of the originally-estimated makespan (and the project buffer is about 33% of the allowed duration). To avoid delays between critical activities that do not share resources, the people in charge of the next critical activity receive an advance warning some time before they can start; otherwise, it is implicitly assumed that release dates for critical activities are unnecessary (unlike the case in the PSB). That is, earliness costs are simply ignored.

Any non-critical activities must have finishing nodes somewhere along the critical path (possibly just ahead of the final finish node), where they merge with the critical path. For instance, in the basic ACM case, all activities merge with the critical path—which consists of the longest expected delivery time in that case—at the finish node. Temporarily, we assume that there is plenty of slack in all these non-critical activities. Because we assume an assembly tree structure, each non-critical activity is the last of a subproject comprising non-critical activities, and each non-critical activity belongs to exactly one subproject. The trick is to treat each subproject as an independent project, with the schedule of the merge point on the critical path acting as its due date. Therefore, this subproject receives a subproject buffer of 50% of its own critical path (again without release dates between activities on the same path), and that buffer is placed just ahead of the merge point. These subproject buffers are called *feeding buffers*. One could also think about them as *assembly buffers*, because they appear just before the output of a non-critical subproject is assembled into the project. Following the same logic, additional feeding buffers may be required within subprojects, wherever an assembly takes place.

Now remove the assumption that there is plenty of slack in all non-critical activities. Because feeding paths often merge with the critical path before the scheduled start of the project buffer, there may not be enough room ahead of the merge event for the full feeding buffer. In such cases, Goldratt (1997)—in a departure from the basic structure given by Pittman (1994)—recommends postponing the start of the original critical path so that the prescribed feeding buffer will be supported. *That is, the critical path is postponed and a feeding buffer now appears in the middle of the longest path*. The motivation for this recommendation is baffling: it requires postponing the project with certainty to avoid the risk that it will be delayed by at most the same amount. Indeed, other CCPM sources also propose an alternative: start such feeding chains together with the critical path but increase the size of the project buffer by the amount "missing" in the feeding buffer (Leach, 2000).[*] Furthermore, the placement of feeding buffers may invalidate the original sequence by requiring activities that rely on the same resource to be performed in parallel (Pittman (1994); Herroelen, Leus & Demeulemeester 2002). Although CCPM recognizes the need to take sequencing conflicts into account explicitly, the use of

with our linear association model, because our model is based on the observation that common causes—such as the same manager—cause linear association.

[*] In the stochastic balance model presented in the chapter, feeding buffers and project buffers are implicit (because we know of no legitimate way to calculate them explicitly). However, we discuss constrained cases where more than one activity should start at time zero. Applying the stochastic balance model where there is no room for a desired feeding buffer could therefore lead to both paths starting at time zero, thus supporting that alternative.

sequencing models of the type we discussed in Chapter 17 is explicitly discouraged (Goldratt, 1997). As a result, the makespan of some CCPM examples in the literature—for instance, see Leach, (2000) and Newbold (1998)—can easily be improved by up to 30%. In this connection, to the extent the flow shop insights of Section 11.5 apply to projects, denser sequences are superior both in terms of mean—including the Jensen gap—and variance.[*]

The buffers specified by CCPM are also used for control. The recommendation is to divide each buffer into three equal bands (associated with the colors green, yellow and red). As the project progresses, buffer consumption is monitored. If the consumption is within the first (green) band, nothing should be done. If the consumption progresses to the second (yellow) band, it is time to plan corrective actions. These corrective actions are actually pursued if the third (red) band is penetrated or exceeded. In contrast to traditional approaches, however, re-scheduling and re-sequencing are not permitted. The buffer-penetration approach provides a clear guideline that is easily understood by the participants and thus helps managers focus on emerging problem areas.

CCPM utilizes a heuristic approach to resolve sequencing conflicts: critical activities come first, and among non-critical activities, the one whose buffer is the most in danger of being consumed receives priority. This heuristic is similar to the minimum-slack priority rule. A similar logic is also presented by Goldratt (1997) to sequence activities that belong to different projects, in which case a critical activity of the project that has consumed a higher percentage of its project buffer receives the highest priority. There is also an explicit CCPM recipe for scheduling multiple projects by Drum-Buffer-Rope (DBR)—which is a scheduling approach associated with TOC. The crux is to identify the resource that is tightest for all projects and sequence as if that resource is a single machine. The buffer-consumption approach is used for that purpose. From the schedule of that resource, working backwards in the MRP fashion, the feeding activities can be scheduled, with standard feeding buffers included. However, this approach can work only if there is a unique bottleneck resource. That assumption is less likely to hold in a chaotic project organization environment than in a flow shop or job shop (Raz, Barnes & Dvir, 2003).[†]

CCPM stresses that Parkinson's Law (which Goldratt renamed *the student syndrome*) applies in practice and causes waste (Gutierrez & Kouvelis, 1991). To ameliorate the Parkinson effect, CCPM requires all activities to be performed at full speed but sets no internal due dates. Such an approach is technically meritorious. However, because it may put pressure on workers constantly, it may not be easy to implement. By contrast, if internal due dates are imposed, explicitly or implicitly (by pressuring the owners of activities that seem to consume too much time), then we can expect that in the next round looser estimates will be given such that after the 50% cut, a hidden buffer remains in the schedule.

Optimal schedules for single-machine models with regular performance measures never require preemption. This feature extends to the case of a single resource type with processing times that are inversely proportional to resource allocation (Portougal, 1972). However, that logic does not necessarily apply to more complex environments, such as job shops and projects with several resource types. The logic also fails if resource allocations should be corrected as part of control. In other words, preemption may be needed because processing times are

[*] The network structure of a flow shop with $m \geq 2$ and $n \geq 3$ is not series parallel, so flow shops with a makespan objective exemplify reasonably complex project structures.

[†] A more robust approach is to control the number of projects that are executed in parallel, and then use priority-based sequencing models of the type we discussed in Chapter 15 on each resource as the need arises.

stochastic. Corrections may also be beneficial when the initial schedule is suboptimal. Ignoring these complexities, a cornerstone of CCPM is to avoid preemption—or multitasking—especially on bottleneck resources. Millhiser & Szmerekovsky (2008) provide a numerical example demonstrating the potential benefit of multitasking even if total processing time is increased as a result. To clarify how beneficial multitasking can be in practice, consider the case of time sharing on mainframe computers. Until the introduction of time sharing, even very short jobs were often queued for a long time. Once the computer started processing them, many of these jobs were rejected almost instantly for some programming error, such as a missing comma. When that happened, the programmer had to fix the error and resubmit the job, which was now liable to have to queue again. With time sharing, very short jobs—including the vast majority of programs with bugs—were processed practically instantaneously; therefore, from the point of view of the human programmers, the debugging process became more efficient by orders of magnitude. Time sharing did waste some time on the mainframe, but overall the effect of multitasking was highly beneficial. Yet, in those days, mainframe time was much more valuable than the time of any individual programmer, so the use of CCPM logic would have subordinated the programmers to the mainframe.

A relatively minor technical point is that the cut estimates used by Goldratt (1997) are medians of processing times rather than means. A typical assumption is that original estimates are about double their median and that they then provide a 90% service level (Newbold, 1998). However, the sum of medians is not the median of the sum; that is, medians are not additive.[*] For instance, if we use the lognormal distribution for individual processing times, and assuming the 90th percentile is twice the median, we obtain $s = \ln(2)/z_{0.9} = \ln(2)/1.28 = 0.541$ (or a coefficient of variation of 0.583). For this s, the median equals 0.864μ. By adding up medians we would thus be underestimating the mean by 13.6%. Therefore, under the traditional independence assumption (which is still accepted by CCPM proponents), the probability of not requiring the buffer is not 50%, as one might surmise (and as CCPM sources actually cite), but a much lower one, approaching zero asymptotically as the number of activities grows. This particular error is academic in the sense that the relevant estimates are not likely to be correct in the first place (because there is no use of history), but it does highlight the general lack of rigor in the CCPM development.

The problems with CCPM are myriad, but we have focused on those that relate to technical scheduling issues. First, project and feeding buffers are important, as was already known before the advent of CCPM, but the CCPM approach for setting them is highly deficient. Buffers should address true data-based stochastic variation whereas CCPM specifies completely arbitrary ones. For instance, if CCPM were applied to the set of nine projects studied by Trietsch et al. (2010), even without first cutting activity times in half, eight out of nine projects would fail to meet their due dates. If the preliminary step of cutting each estimate in half had been taken, then none of the projects would complete on time (although we can't tell whether other CCPM practices would have ameliorated the problem). Furthermore, buffers should not be planned explicitly as in CCPM but rather determined implicitly by optimizing release dates, as in the PSB model.[†] Second, sequencing algorithms (including effective heuristics) are not actually used, and

[*] Medians are additive when the distributions are symmetric. But the typical CCPM assumption is that realistic processing time distributions tend to be skewed to the right. Indeed, Newbold (1998) is one of few sources that recommend the lognormal distribution for processing times.

[†] On the bright side, the arbitrary approach of CCPM avoids the classical PERT pitfall of setting relatively negligible buffers for projects with many activities (Trietsch, 2005b). We do not know of a CCPM source that lists stochastic

thus major improvement opportunities are often squandered. Third, the buffer-setting method may invalidate the informal resource allocation that is used to identify the critical path (or "critical chain"). Instead, the approach of the chapter should be adopted: use soft precedence constraints to enforce all sequencing decisions. The excessive focus on binding constraints is simply flawed and, to our knowledge, has never been validated in practice. Finally, the perplexing recommendation to postpone the critical path if a feeding buffer is too small and the incorrect usage of medians both indicate lack of technical professionalism in the CCPM development by Goldratt (1997).

In spite of these issues, CCPM has received a lot of attention among project management professionals and academics. That attention highlights the fact that practitioners are not satisfied with PERT/CPM *as practiced*. Thus, the advent of CCPM helped focus renewed research interest in PERT/CPM. Trietsch (2005) lists research questions motivated by CCPM; for instance, how to pursue sequencing, buffering and crashing in a stochastic environment, and how to actually estimate the necessary distributions. By now, we have made several contributions along these lines that are already reflected in the text. Recall that we refer to the new framework as PERT 21 (see the Research Notes for Chapter 16). As such, PERT 21 owes much of its motivation to CCPM. But there are major technical differences between the two approaches. PERT 21 is based on sound stochastic analysis, utilizes information fully, and, by the balance principle, assures economic responses to developments, whereas CCPM is lacking on all those fronts. In terms of control, that is probably CCPM's strongest suite, but the PERT 21 control method is quite different to the buffer monitoring of CCPM. First, it is not buffer consumption that really matters but rather the projected effect on the objective function. Indeed, for the same reason that we cannot size buffers explicitly (opting for setting release dates instead), we can also not monitor them explicitly. More precisely, buffer consumption is not directly related to the truly important measurement, which is criticality. In PERT 21, by reassessing plans based on current information (contrary to the CCPM recommendation), we can easily assess criticalities directly on an ongoing basis. Next, the action limits CCPM recommends (dividing the buffer to three bands) are not relevant to statistical control; that is, they are not data based. In our framework, although we do not formally identify "in-control" and "out of control" situations, we have two types of response: rescheduling—which suits predictable deviations—and re-planning (including re-sequencing and expediting)—which suits unpredictable shocks and design changes. Finally, for control purposes, CCPM only monitors consuming more time than expected, while better-than-expected performance is ignored. It is tempting to say that if things go better than expected we should leave well-enough alone, but there may be situations in which some release dates have to be advanced due to earliness in previous activities. Or we might be able to opt for less expensive modes (that is, use negative crashing).

It is also important to discuss focusing and hierarchy. Under PERT, the focus is on the critical path, and the same focus is the key to CCPM (although during the control phase, CCPM focuses by buffer penetration regardless of whether it is on the critical path). Under stochastic economic balance, the whole concept of "critical path" becomes much less relevant. Indeed, the concept is rooted in deterministic thinking! Instead, at any stage of project execution, the activities in process constitute a cut set for the project network, and they all have varying degrees of criticality (such that the sum of criticalities in the cut set is unity). Focusing should utilize the

dependence as a reason for that recommendation, however. By contrast, Leach (2003)—perhaps the most solid explanation of the CCPM buffer setting approach—lists eleven *other* reasons (such as the Jensen gap and the Parkinson effect).

criticality information, and because there may be many activities in a cut set, it is necessary to focus on the most critical activities or the ones most out of balance by the Pareto principle. However, if top management focuses on some activities, the others must be delegated. Arguably, the connection between the delegated activities and the rest of the project should then be buffered so that they can be managed with an adequate degree of autonomy (Beer, 1981). PERT 21 does not dictate such "autonomy buffers," but it enables their inclusion in a balanced way. Furthermore, since delegated activities rarely possess high criticality, they are likely to be adequately buffered automatically. Regardless, the predictive Gantt chart provides useful information to all stakeholders by showing the status of the project, the criticality of each activity, and its interaction with other activities. Furthermore, the most critical activities may be scheduled in the future, and an important central management focus is managing future activities. For example, as mentioned above, it may be necessary to decide whether to expedite some activity on the current (or forthcoming) cut set, or wait for a later stage and expedite then. Doing this well is still an art at this stage, and requires further research. But PERT 21 can support such decisions. In addition, in PERT 21 we can adopt the CPM hierarchical approach where an activity at a high hierarchical level may represent a subproject at a lower hierarchical level. For instance, consider the predictive Gantt charts of Figures 19.5 and 19.6, where the bottom "project" level encompasses the combination of the top five levels. That bottom level could act as a single activity in a higher hierarchical level. (See also Section 19.3, where we discuss hierarchy.) Such a hierarchical structure is imperative in large projects. By contrast, CCPM assumes that all activities are explicit.

One of the biggest attractions of CCPM (and of CPM) is that it requires only single-point estimates for activities. One of its most distressing weaknesses is that these estimates are then used in an arbitrary manner and cannot be appropriate in general. In PERT 21 we also require only single point estimates, but we enhance them with historical regression analysis. By incorporating valid stochastic models for sequencing, crashing and control, PERT 21 combines the strengths of traditional PERT/CPM with the simple user interface of CCPM without succumbing to the major theoretical weaknesses associated with both these methodologies.

Part 2: Alternative Balancing Frameworks

The coverage of hierarchical scheduling in Section 19.3 was expanded in the second edition: in the first edition it was limited to the current Example 19.5. That is, the material about balancing subprojects with progress payments is new in the second edition. A relevant source that we were not aware of by the time the book went to print is Jansen et al. (2018)—see also Jansen (2019)—who presented a model with progress payments that also allows coordinating separate projects. The hierarchical balancing framework in our text, including the treatment of progress payments, was derived independently. As for Example 19.5, it can also be used to illustrate a different balancing framework that is likely to be appropriate for small projects whose main inputs are standard off-the-shelf components. If such inputs, and perhaps general labor costs, largely determine the real holding costs that we should account for, then holding costs only apply when we actually acquire the inputs, not necessarily as early as the release dates for those acquisitions; that is, if for some reason an activity start is delayed beyond its release date, procurement can be postponed and therefore the holding cost is also postponed. For instance, in Example 19.5, suppose that another supplier offers trucks and cranes for hire without having to book them in advance. Likely, such a supplier controls enough trucks and cranes to meet demand with a sufficiently high service level, but what counts for our example is that instead of booking the resources before we actually know precisely when we can use them, we can hire them when the time comes without booking, albeit, likely, for a higher hourly charge (and a different delay distribution). That option is equivalent to off-the-shelf purchasing because it entails holding costs that only start when we actually start the activity. We refer to such a cost structure as *Pay as Realized* (PAR). By contrast, the model we introduced in the chapter assumes that activities start carrying holding costs at their planned start, a cost structure to which we refer as *Pay as Planned* (PAP). PAP is especially valid when activities require advance preparations that make it impossible to delay cost. For example, that is often the case when an activity is actually a subproject. By contrast, PAR is likely to be appropriate when activities can be started at short notice, including the case of some small projects or subprojects. It would be an error, however, to think that small projects always justify PAR. For instance, consider Example 19.3 where we applied PAP to a bus route. Essentially, we treated setting the bus schedule as a (very small) project. But because passengers arrive based on the published schedule, they incur holding costs as soon as the scheduled release date occurs, regardless of whether the bus is on time or not. Hence, PAP is indeed appropriate in that case. Historically, PAR models have their roots in the mid-1980s (Yano 1987, 1987a), which is also the case with PAP, but the two models were pursued independently of each other until recently (Jansen et al. 2019, and see also Jansen 2019). In what follows we first cover insights from Jansen (2019), but we do so in terms of sample-based analysis. Recall from the text that sample-based analysis facilitates the practical solution of stochastic models that do not rely on assumptions such as statistical independence or particularly convenient distributions. Indeed, in this case too, the use of samples allows us to provide solid engineering solutions for more complex models than those originally solved by Jansen (2019). We show how to find local minima, and because the objective function tends to be flat near optimum, such local minima are likely to be fully satisfactory from a practical point of view.

Although Jansen (2019) provides the basis of our coverage, we use the notation and terminology of the text where possible, but we use Activity on Node (AON) rather than Activity on Arrow (AOA). As in the text, we use C for completion time, p for processing time (duration)—both with appropriate indices when required—and $P(j)$ for the set of immediate predecessors of activity j. Unless specified otherwise, activities are numbered from 1 to n such that if j is a successor of i then $j > i$. Under both PAR and PAP, activity j cannot start before its release date, r_j. By these rules, activity 1 must be a root activity and activity n cannot have a physical successor, but for notational convenience we refer to *strict* tardiness (if any) as a pseudo-activity with the index $n + 1$ and (as in the text), the "release date" of tardiness, r_{n+1}, is the project due date. We may also denote strict tardiness by T; that is, $\mathrm{T} = n + 1$ so $r_T = r_{n+1} = d$. We say that an activity *breaches* a subsequent release date—including r_{n+1}—if it strictly delays the start of the subsequent activity. More formally, we should say that in a *particular scenario* the activity breaches a subsequent one, but we take the liberty of using terms like scenario, activity and even release date interchangeably when the context is clear; for instance, we may say a scenario breaches a release date but more formally we should say that an activity or chain of activities in this scenario does that. Whereas breaching always involves *strict* delays, weak tardiness and weak criticality are special cases of tardiness and criticality (as in the text), even though they do not involve breaching the due date. If two or more parallel activities exceed a common successor release date by different amounts, only the one that exceeds it most—and is thus the one that actually delays the successor—is considered to breach it. (In the unlikely event that two or more parallel activities exceed a common subsequent release date by the same amount, we can choose arbitrarily which one to consider as the breaching activity.) In the serial case, when an activity exceeds the gap between its release date and the next release date, breaching must occur; but it can also occur due to a sufficiently large upstream breach. A contiguous set of breaching activities together with the last breached activity is called a *block*. For instance, if activity n starts at r_n and breaches the due date, r_{n+1}, we get a block of two activities, *n* and *n* + 1 (or *n* and T). However, we consider a single activity that starts at its release date and does not breach any other activity as a block too, albeit a *degenerate* one. By this definition, the last activity in a block does *not* breach any successor. For instance, if an activity ends precisely at the release date of its only successor, it is last in its block (because it does not strictly delay the next activity). In particular, if activity n starts at r_n and is weakly tardy, it forms a degenerate block (and the scenario does not include tardiness as a pseudo-activity). Emphatically, whereas blocks often comprise a serial chain of activities (which is always the case in the serial case), that is not necessarily the case in general; we present counterexamples later. We reserve the terms *independence* (or *dependence*) and *independent* (or *dependent*) to imply that activity durations are statistically independent (or statistically dependent). As in the text, α_j is the holding cost of activity *j*, $\beta = \alpha_{n+1}$ is the tardiness penalty, and $\sum_{j=1}^{n} \alpha_j = \alpha$. For convenience, without loss of generality, we set $\alpha + \beta = 1$ for each scenario (which may require scaling when setting up the model and scaling back when calculating the costs). Accordingly, for PAP, α_j can also be interpreted as a target probability to be *matched* by the model (for *j* = 1, …, *n*, *n*+1). (When using a sample, matching a target probability requires achieving the desired target as closely as sample-based analysis allows. It is akin to using the inverse cumulative distribution function [cdf] of a continuous random variable. Formally speaking, such an inverse does not exist for a sample-based empirical cdf. Nonetheless, Figure B.1, in Appendix B, illustrates how we can match a desired probability when the cdf is a step function.) As a default, we assume *d* is sufficiently large to render all *n* optimal release dates strictly positive. However, solving constrained cases that violate the assumption that *d* is sufficiently large under PAR is straightforward, and follows the treatment of constrained cases under PAP: if balancing a release date requires a negative release date, the optimal release date is zero (and the desired service level may not be achieved). In Chapter 19 we presented the PAP objective function for a stored sample as a linear program. Quoting from there, with minor notational changes and other edits in brackets:

> The model is essentially an elaboration of the generic [Project Stochastic Balance] problem, but instead of using the expected value in the objective function, we use the average cost computed for the scenarios in the sample. For each scenario, we use its own processing time realizations, but the same release date decisions apply to all *S* scenarios. The project completion time of scenario *s* is denoted C_s. For other variables, we use a double index (*s j*) to denote the *j*th activity of the *s*th scenario. We obtain
>
> $$\text{Minimize Z} = \frac{(\alpha+\beta)}{S}\sum_{s=1}^{S} C_s - \sum_{j=1}^{n+1} \alpha_j r_j$$
>
> Subject to
>
> $$C_{sj} \geq r_j + p_{sj};\ s = 1, \dots, S, j = 1, \dots, n$$
> $$C_{sj} \geq C_{sk} + p_{sj};\ \forall k \in P(j), s = 1, \dots, S, j = 1, \dots, n$$
> $$C_s \geq d = r_{n+1}$$
> $$C_s \geq C_{sk}; \forall k \in P(N), s = 1, \dots, S$$

> We can solve this model as a generic linear program. In practice, however, it is more efficient to find the optimal solution by a numerical search […]. For a sample with an appropriate number of scenarios, the linear programming (LP) formulation is unwieldy even when the project is small. It is important mainly because it implies that the problem is convex, so once a local optimum is found by such a search, we know that it is globally optimal.

This excerpt applies to AOA networks, but activities are identified by single indices, rather than as pairs of nodes (as is the usual practice under AOA). However, N denotes the final node of the project network rather than the last activity in numerical order (n). Hence, $P(N)$ is the set of all activities followed directly by d (and n is in this set), whereas $P(j)$—including the case $j = n$—is the set of immediate predecessors of activity j. The text also clarifies that dummy predecessors have to be replaced by their own physical predecessors, but that is not necessary for AON. Here we adopt AON, so we have to remove N and edit the conditions accordingly. Before we do that, however, note that these conditions can be rewritten as maxima. Furthermore, because we are minimizing $\sum C_s$, there cannot be an incentive to exceed these maxima; therefore, once rewritten as maxima, the conditions are guaranteed to be satisfied as equalities. Hence, except for the calibration necessary for setting $\alpha + \beta = 1$ and editing the conditions for AON notation, the following model is identical:

$$\text{Minimize Z} = \sum_{s=1}^{S} C_s - S \sum_{j=1}^{n+1} \alpha_j r_j$$

Subject to

$$C_{sj} = \max_{k \in P(j)} \{r_j, C_{sk}\} + p_{sj};\ s = 1, \dots, S, j = 1, \dots, n$$
$$C_s = \max\{d, \max_{j=1,\dots,n} \{C_{sj}\}\};\ s = 1, \dots, S$$

Under PAR, however, the model is not convex so it is impossible to formulate it as an LP. But we can still use the same conditions (because they are now presented as equalities). Altogether, the following mathematical program applies for PAR:

$$\text{Minimize Z} = \sum_{s=1}^{S} C_s - \sum_{s=1}^{S} \sum_{j=1}^{n} \alpha_j (C_{sj} - p_{sj}) - S\beta d$$

Subject to

$$C_{sj} = \max_{k \in P(j)} \{r_j, C_{sk}\} + p_{sj};\ s = 1, \dots, S, j = 1, \dots, n$$
$$C_s = \max\{d, \max_{j=1,\dots,n} \{C_{sj}\}\};\ s = 1, \dots, S$$

Focusing on the objective function, $(C_{sj} - p_{sj}) \geq r_j$ is the start time of activity j under scenario s, and we subtract $S\beta d = S\alpha_{n+1} r_{n+1}$ because the tardiness pseudo-activity, if any, starts at d. Compared to PAP, where we used r_j instead of $(C_{sj} - p_{sj})$, the objective function of PAR is bounded from above by that of PAP for any given set of release dates. That should not be surprising given that PAR postpones payment relative to PAP without delaying the project. By optimizing the release dates for PAR, its advantage over PAP can only increase. One heuristic approach follows this observation: start by solving the PAP model and then search locally for an improvement. But we can do better than that, and in some independent instances have an optimality guarantee. (For future reference, it is straightforward to generalize this mathematical program for the mixed model where some activities are PAP and some are PAR. In that context, even in "pure" PAR instances, root nodes are always PAP: they cannot be delayed by predecessors so $\left(C_{sj} - p_{sj}\right) = r_j$ for them.)

It can be shown that the optimal project service level, SL, is the same for PAP and PAR models (in particular, for $\alpha + \beta = 1$, $SL = \beta$). Furthermore, under independence, if SL is close to 1, the difference between the models tends to be small. The distinction is more important if SL is low or if activities are positively correlated (for instance, under linear association). When compared to PAP, PAR upstream release dates acquire higher criticalities at the expense of downstream activities and the total cost is reduced. That is because when a non-degenerate block starts at an upstream release date, this release date controls the costs of all the downstream activities of the block (including strict tardiness, if any); with that control comes a "criticality allowance," and thus the upstream release date collects, or inherits, a higher criticality than that accorded by the first block activity alone. We elaborate later.

Jansen (2019) solves PAR for converging project networks under independence. Converging networks are series-parallel networks where no activity is followed directly by more than one other activity (but assemblies are allowed). The Assembly Coordination Model (ACM) of Section 19.2.1—where we coordinate the release dates of n parallel activities preceding a common due date—is an example of a converging network. Notably, it has the same optimal solution under PAR or PAP (because all n activities are root nodes). But a converging network with as few as two parallel activities followed in series by a single stochastic assembly activity (Yano, 1987a) is different under PAP and under PAR. A special case of a converging network is a serial project under independence (Yano, 1987). Below, we apply sample-based analysis to the solutions of Jansen (2019). We start with a one-pass algorithm for the serial independent case. Next, we show that adapting the algorithm to general converging networks requires iterations, which is not surprising when we recall that the ACM requires iterations (and the ACM is a special case of a converging network). We then present another sample-based one-pass algorithm, also based on Jansen (2019), and show that the two algorithms are equivalent (although they respond to sampling errors differently). The latter algorithm, which matches the gradient to zero explicitly, provides the basis for our unpublished extensions. These include dependence, more general networks, and mixed models that allow both PAP and PAR. However, we do not offer proof of global optimality for most of these extensions. One problem with relying on setting (or matching) the gradient to zero is that, under PAR, the problem is not convex. Furthermore, typically there are at least two distinct zero-gradient solutions, although they may merge with each other in some special cases. As we show later, one of those is obtained when all serial release dates are set to the same value, which is patently suboptimal in most cases. More precisely, it is suboptimal unless it is the only zero-gradient solution. In what follows, as a default, we ignore the existence of this suboptimal solution. Furthermore, we only present algorithms that avoid it when it is distinct.

Before presenting the one-pass special case solution, we elaborate on matching and we emphasize an important characteristic of using a stored sample (with S runs). As S grows large, the stored sample becomes a progressively more reliable representation of the distribution used to generate it. In Appendix B and elsewhere in the text, we match a desired criticality of f (which is equivalent to a service level of 1 – f) by sorting the sample and selecting the $\lceil S(1-f)\rceil^{th}$ smallest scenario to determine the tightness of the desired release date (where $\lceil x \rceil$, the ceiling function, is the smallest integer that is at least as large as x); that is, we allow just enough time for this scenario. This is equivalent to using the empirical cumulative distribution function (cdf) the sample provides with added vertical segments connecting discrete steps, where matching involves selecting one of the x-axis arguments of these vertical segments (Figure B.1), and if it falls precisely at the level of a step, we use the sample value associated with that step, which is at the left edge of the step (such that the vertical segment is not above the step). Scenarios with strictly higher durations are strictly tardy or strictly critical; scenarios with strictly lower durations are strictly early or loose; the $\lceil S(1-f)\rceil^{th}$ smallest scenario itself is both weakly tardy and weakly early, and the same applies to other scenarios with the same duration, if any. Therefore, at least $\lceil S(1-f)\rceil$ scenarios are strictly- or weakly early, and, equivalently, at least $\lceil Sf \rceil$ scenarios are strictly- or weakly tardy. Furthermore, at least one scenario is both weakly early and weakly tardy, and we may also call it *neutral*. The objective function is piecewise linear so the derivative—which is a step function—is not properly defined at some points, including any point that is associated with a neutral scenario; but we can calculate shifted derivatives for such a point by employing negative- or positive infinitesimal shifts. When employing a negative shift, we refer to the derivative as the *left-derivative*, and a positive shift yields the *right-derivative*. It can be shown that when allowing for the $\lceil S(1-f)\rceil^{th}$ smallest scenario the left-derivative is strictly negative and the right-derivative is nonnegative. The difference between the two sides is because neutral scenarios count as strictly early to the left (shifting the release date to the left increases the time allowance for all scenarios) but tardy to the right (at least weakly). The uniqueness of the solution depends on whether the right-derivative is strictly positive, which is the case if $\lceil S(1-f)\rceil > S(1-f)$; if $\lceil S(1-f)\rceil = S(1-f)$ then there is a flat segment to the right, the right derivative is zero, and the optimal solution is not unique. In what follows, for convenience and without loss of optimality, we ignore the possibility that the solution is not unique; that is, if multiple optima exist, we use the one identified by matching the $\lceil S(1-f)\rceil^{th}$ smallest scenario (for which the left derivative is strictly negative). Now assume independence and suppose we were to decrease the sample size by removing some randomly selected scenarios. Because the removed scenarios are selected randomly, the remaining reduced sample is still random. If we re-solve the problem for the reduced sample, the best release date may change; for instance, the particular scenario that was selected to be neutral before may have been removed. But, under independence, the expected value of the optimal release date should not change. Because we know that scenarios were removed *randomly*, we have the option of *not* updating the previous solution even if it is no longer best for the reduced sample: it is still based on a legitimate (and larger) sample and therefore it is still valid and, arguably, preferable. Henceforth, we refer to that as the *larger sample validity principle*.

Now consider how we might solve for a serial project under PAR. Because we assume d is large, we know that the final solution will only involve strictly positive release dates and hence we can find them by starting at 0 and tightening. (If that assumption is incorrect, 0 may be the optimal value for at least some release dates.) With that in mind, we can now present an algorithm based on Jansen (2019), but using a stored sample. The algorithm proceeds backwards, setting r_n first. One pass is sufficient under independence.

Algorithm RN19.1

Setting Release Dates for Serial Projects under PAR

```
Step 1. Initially, let $k = 0$ and let $r_j = 0$ for $j = 1, \ldots, n$.
Step 2. Ignoring all upstream activities, postpone (that is, increase) $r_{n-k}$ incrementally (thus
        tightening $d - r_{n-k}$) just enough so that at least a total of $\lceil S \sum_{j=n-k}^{n} \alpha_j \rceil$ scenarios are
        critical (that is, match or exceed the due date).
Step 3. If $k < n - 1$, increment $k$ by 1 and return to Step 2. Else, stop.
```

In a nutshell, under independence, the algorithm is optimal because it is equivalent to matching the gradient of the objective function to zero in a neighborhood where small perturbations are detrimental and where there is only one such solution. We repeat, however, that under PAR we can typically construct a suboptimal solution with a zero gradient, but that solution, if different (which is the typical case), is not in a neighborhood where small perturbations are detrimental. Another algorithm, presented later and denoted Algorithm RN19.2, matches the gradient to zero directly and in the same neighborhood. After presenting the new algorithm we show that the two algorithms provide two valid ways to match the gradient to zero in a neighborhood where small perturbations are detrimental, and hence they are equivalent. However, Algorithm RN19.1 relies on the larger sample validity principle and Algorithm RN19.2 does not, so they respond to sampling errors differently and the solutions they provide are not identical.

For now, it is useful to study Algorithm RN19.1 in more detail. For $k = 0$, it is straightforward to match the required tardiness of $\lceil S\alpha_n \rceil$ scenarios, because a scenario, s, counts as tardy for $k = 0$ if and only if $d - r_n \leq p_{sn}$. But for upstream activities ($k > 0$), more than $\lceil S \sum_{j=n-k}^{n} \alpha_j \rceil - \lceil S \sum_{j=n-k+1}^{n} \alpha_j \rceil$ scenarios may breach the next release date, r_{n-k+1}, before we achieve the desired number of *new* tardy scenarios: some just increase the tardiness of scenarios that have already been tardy in previous iterations, and some may breach the next release date, r_{n-k+1}, without the block thus created breaching (or even matching) the due date. For the same reason, the optimal criticality of upstream release dates can exceed that of PAP, at the expense of downstream release dates that are now breached.

Notably, the algorithm does not always yield upstream release dates that are earlier than downstream ones. That is, as we tighten r_{n-k} for some $k \geq 1$ we may not yet achieve the required total criticality when $r_{n-k} = r_{n-k+1}$ and thus the final result can be that $r_{n-k} > r_{n-k+1}$ (which renders r_{n-k+1} irrelevant); in such case we say that r_{n-k} *dominates* r_{n-k+1}. Under PAR that may be optimal: it just means that r_{n-k+1} is never *active* (that is, it does not constrain the start time for any scenario), so activity $n - k + 1$ always starts as soon as activity $n - k$ completes. Examples are not very difficult to construct—particularly when the mean and variance of the downstream activity are relatively large (see Example RN19.1 below). Operationally, we can reset those downstream release dates to coincide with the tighter upstream ones, but there is no compelling reason to do so. As a default we assume no such resetting is applied. Under PAP, however, $r_{n-k} > r_{n-k+1}$ is impossible in an optimal solution (Jansen, 2019): that is true because when $r_{n-k} > r_{n-k+1}$ then r_{n-k+1} is no longer critical for *any* scenario whereas an optimal PAP release date must be critical with a frequency of α_j at least. Note also the restriction to $k \geq 1$: even under PAR all optimal release dates are strictly smaller (that is, earlier) than the due date. Last but not least, it is possible for r_1 to dominate all the downstream release dates, in which case there is only one zero-gradient solution and this solution is optimal. (Because we do not reset dominated release dates, however, we consider the solution provided by Algorithm RN19.1 in such case as distinct from an operationally equivalent solution where all release dates are set equal to r_1. This distinction is important because in general there is a zero-gradient solution where all due dates are constrained to be equal in advance, but that solution is only optimal if Algorithm RN19.1 provides a solution where r_1 dominates all other release dates. In such case, the Algorithm RN19.1 solution is practically guaranteed to be distinct; that is, other release dates generated by it will be strictly smaller than r_1. We discuss this issue in more detail later.)

Example RN19.1. Consider a serial project with 3 input activities and a due date of $d = 200$. The durations of each activity are independent and follow a lognormal distribution with *cv* = 1 and means $\mu_1 = 50$, $\mu_2 = 10$, and $\mu_3 = 100$. Earliness costs (α_j) per scenario are 0.07, 0.14, and 0.49, respectively. Hence the tardiness cost per scenario is $\beta = 0.3$.

To find the optimal release dates for this project, we use a sample of 1001 scenarios (which is available on the book site). For this particular sample, if we set $r_3 = 126.8421$, with the other release dates at 0, then 491 scenarios are tardy and 511 are early (with one neutral scenario counted twice). Hence, this value is the solution for k = 0. For k = 1, we find that for $r_2 =$ 134.2488 there are 631 tardy scenarios (an increase of 140, commensurate with $\alpha_2 = 0.14$) and 371 early ones. Because 134.2488 > 126.8421, it is clear that r_3 is dominated by r_2. Next, for $r_1 = 69.8741$ we obtain 701 tardy scenarios (an increase of 70, commensurate with $\alpha_3 = 0.07$) and 301 early ones, indicating that it is the optimal r_1. The objective function value is 106,980.8.

If we only had activities 2 and 3 to consider, then, effectively β = 0.37, r_2 and r_3 would still be optimal at 134.2488 and 126.8421, and the objective function value would be 89,278.8. The next example utilizes this observation to compare PAR to PAP.

Example RN19.2. Consider a serial project under PAP with 2 independent activities, indexed 2 and 3, and a due date of $d = 200$. As in Example RN19.1, the durations of each activity are independent and follow a lognormal distribution with *cv* = 1 and means $\mu_2 = 10$ and $\mu_3 = 100$. Earliness costs (α_j) per scenario are, again, 0.14 and 0.49, respectively. Hence the tardiness cost per scenario is $\beta = 0.37$.

Again we use the same sample of 1001 scenarios, but we ignore the data for activity 1. The solution now requires iterations but we can choose to work backwards as in Algorithm RN19.1, with recursion, albeit with a different balance criterion (as presented in Chapter 19): under PAP, it is in the *final* solution that r_j should have $\lceil S\alpha_j \rceil$ critical scenarios starting at it (whereas under the original Algorithm RN19.1 that is the desired additional net criticality at stage $k = n - j$). In other words, unlike the independent PAR case, critical scenarios that are subsequently breached from above must be compensated for, iteratively. Here, the final PAP solution is obtained with r_2 = 128.7867 and r_3 = 143.2616 (compare to 134.2488 and 126.8421, respectively). The objective function value is 91,026.53. Recall that under PAR, if we ignore activity 1, the optimal objective function value is 89,278.80, so the increase is almost 2%. If we test the objective function value under PAR but with the PAP optimal release dates, it is 90,093.86, a penalty of about 0.9%; hence, in this example, more than half of the PAR advantage is obtained even if we do not change the PAP solution. Finally, even though the service level is not high, the penalty of solving for PAP when the real costs are PAR is not dramatic. Nonetheless, the release dates are sizably apart under the two versions. The smallness of the difference is not surprising because the objective function of unconstrained balance models is typically quite flat near the optimum.

Example RN19.3. Consider a serial project under PAR with 2 independent activities, indexed 1 and 3, and a due date of $d = 200$. As in Example RN19.1, the durations of each activity are independent and follow a lognormal distribution with *cv* = 1 and means $\mu_1 = 50$ and $\mu_3 = 100$. Earliness costs (α_j) per scenario are, again, 0.07 and 0.49, respectively. Hence the tardiness cost per scenario is $\beta = 0.44$.

This example is similar to the previous one but we ignore activity 2 instead of activity 3 and we solve for PAR. The results are r_3 = 126.8241 (as per the original result for this release date) and r_1 = 60.0911. Notably, 60.0911 < 69.8741 so, perhaps counter-intuitively, by removing activity 2 from the picture we now have to start activity 1 *earlier*. That is the case because in the full solution of both previous PAR cases we delayed r_3 to match r_2 and, in addition, *SL* is higher here.

Even under independence, a single pass of Algorithm RN19.1 only solves the serial project case. The full solution for any converging network under independence, as provided by Jansen (2019), cannot be achieved by any known one-pass algorithm. But it can be solved by running Algorithm RN19.1 iteratively, correcting some release dates while holding some other release dates constant, as per the last iteration. To provide a useful example, the simple network with two parallel activities followed in series by a single assembly activity that was analyzed by Yano (1987a) is a good start. Whereas this simple Y-shape case is sufficient to show that iterations are required—in fact even an ACM with two activities, a V-shape case (without the assembly activity or with a deterministic assembly activity) is sufficient for that purpose—we consider a slightly more complex Y-shape case where each activity is replaced by a serial chain of two or more activities, so the assembly activity—which we also refer to as activity A—is followed by one or more activities in series, and similarly at least one activity follows each root in series before A is reached. Recall that activities are indexed in technologically feasible sequence (that is, if i is a predecessor of j then $i < j$). Therefore, we can implement Algorithm RN19.1 directly, albeit as a heuristic (which is true regardless of network structure). In our example, if there are $m \geq 1$ activities following A, then the last $m + 1$ activities must be indexed between $n - m$ or A (for the assembly) and n, but there is some flexibility in indexing the other $n - m - 1$ activities connecting the two roots to the assembly. We know, however, that one of the two roots must be indexed as 1. We refer to the serial chain of activities leading from activity 1 towards A (but not including A) as Chain 1. Similarly, Chain 2 leads from the other root towards the assembly. The remaining activities (the stem of the Y-shape), starting with A (or $n - m$) and including activities $n - m + 1, \ldots, n$, and possibly $n + 1$, form Chain 3. When we apply Algorithm RN19.1, release dates are set for Chain 3 before any other activities are considered. These particular release dates, including the assembly release date, r_A, are optimal after the first pass (we elaborate later). But all the remaining $n - m - 1$ release dates, in Chains 1 and 2, will require multiple passes. For instance, suppose that after setting r_A we apply Algorithm RN19.1 to Chain 2. If so, for some scenarios, blocks will be created that start within Chain 2 and end within Chain 3 (and all of them breach r_A); we refer to such blocks as *2-to-3* blocks (and a block connecting Chain 1 to Chain 3 would be *1-to-3*). Next, we set the release dates for Chain 1. Each one of those release dates can create 1-to-3 blocks that breach 2-to-3 blocks, including 2-to-3 blocks that were previously counted as new for the purpose of satisfying Step 2 of Algorithm RN19.1. Such breaching occurs if a new 1-to-3 block breaches r_A by more than a previous 2-to-3 block did. The previous 2-to-3 block no longer breaches r_A, so now it terminates just above the assembly. The assembly itself and any activities downstream that may have been part of the original 2-to-3 block now become part of the new 1-to-3 block, and activities further downstream may also be appended if the breaching of r_A is enlarged sufficiently (it is always larger than before). When that happens for at least one scenario, the previous settings of the activities at the heads of all such original 2-to-3 blocks are no longer valid. In response, previously set release dates have to be reset, leading to iterations. During such iterations, when rebalancing release dates in Chain 2 (Chain 1), we keep the release dates of Chain 1 (Chain 2) constant but, following the algorithm, we ignore any breaching from upstream; that is, we work from the assembly backwards towards the root in the same way as Algorithm RN19.1 would require if applied to that chain alone. Generalizing the insights obtained by this relatively simple example, any assembly node requires iterations to make sure that the criticalities of all activities leading to it and to any downstream assembly are correct.

As we elaborate presently, there is a direct relationship between block structure and the partial derivatives of the objective function by the release dates; for unconstrained release dates, balance implies that the gradient is matched to zero, and changing blocks undermines that match. The relationship between blocks and the gradient is also the basis of our proposed engineering solution for more general models. These models generalize the PAR independent converging network model by allowing dependence, considering general project networks (for instance ones that incorporate the interdictive graph), and allowing mixtures of PAP and PAR activities. However, it is not straightforward to adapt Algorithm RN19.1 for such models. That is, we need a way to judge whether balance has been achieved and, if necessary, how to correct for lack of balance downstream after upstream adjustments affect it. For this purpose, instead of counting new critical scenarios while ignoring breaches from above—the search criterion of Algorithm RN19.1—we propose matching the partial derivatives by the n release dates to zero directly. Henceforth, for brevity and where the context is clear, we may refer to partial derivatives simply as derivatives and we may omit the qualification "to zero." By matching derivatives we mean finding solutions where left derivatives are negative and right derivatives are nonnegative, generalizing our observations about left- and right derivatives for a single activity earlier. Matching derivatives is applicable to the proposed generalizations. However, matching the gradient to zero only guarantees local minima, so we have to avoid patently inferior solutions. If we do that, and considering that all these models are relatively flat near optimum, we obtain satisfactory solutions for engineering purposes (aka practical purposes).

Algorithm RN19.2

Setting Release Dates for Serial Projects under PAR

```
Step 1. Initially, let $k$ = 0 and let $r_j = 0$ for $j$ = 1, …, $n$.
Step 2. Ignoring all upstream activities, adjust $r_{n-k}$ to the lowest possible value necessary to
        render the right partial derivative by $r_{n-k}$ nonnegative (for that value, the left derivative
        is negative).
Step 3. If $k < n - 1$, increment $k$ by 1 and return to Step 2. Else, continue.
Step 4. Calculate the objective function value and if it is sufficiently close to an available
        previous value, stop. Else, set $k$ = 0 and return to Step 2.
```

Under independence, one pass of Algorithm RN19.2 would suffice, but we present it with iterations because we do not assume independence. Almost needless to say, in Step 4 we could set other stopping criteria, such as requiring the gradient to be matched to zero perfectly or restricting the number of iterations. When we address the more general converging case, we implicitly assume that Algorithm RN19.1 is also adapted to allow iterations. To continue, we now show that Algorithm RN19.1 matches the expected value of all partial derivatives to zero and that it is guaranteed to yield the unique optimal solution for independent durations. As a byproduct, by doing that we can also demonstrate that Algorithm RN19.2 converges to the same optimal solution (albeit subject to different sampling error effects). But first, we cover some background material and introduce further notation.

As long as we restrict ourselves to converging networks (and serial networks are included as a special case), blocks are always serial. That is true because there are no divergences in a converging network. For a block associated with scenario s and starting at r_j, denote the negative sum of the holding costs in it by v_{sj}, but set $\alpha_{n+1} = -1$. Therefore, if the block terminates at activity m, then, for the serial case, $v_{sj} = -\sum_{i=j}^{m} \alpha_i$. If $m \leq n$, then $v_{sj} < 0$; if $m = n + 1$ then $v_{sj} > 0$, and if so we can also write $v_{sj} = 1 - \sum_{i=j}^{n} \alpha_i$. If r_j is breached from above in scenario s, we write $v_{sj} = 0$ (because for that scenario the upstream release date at the head of the block controls the start and finish times of activity j and any activities downstream in the same block). Until further notice, assume the block we focus on, starting at r_j, does not complete precisely at the release date of the following activity or, if $m = n$, the due date. Similarly assume that no upstream block matches r_j precisely. If we increase r_j by an infinitesimal positive amount, ε, we have to add εv_{sj} to the objective function (because less safety time is invoked), and this value is positive if and only if strict tardiness is the last activity in the block. It follows that scenario s contributes v_{sj} to the right derivative by r_j. We denote the contribution of the sth scenario to the right partial derivative by r_j as v_{sj}^{+}, so under this assumption we can write $v_{sj}^{+} = v_{sj}$. Now remove the assumption that the block does not match the next release date. If the block matches the following release date (including the due date, r_{n+1}), then calculating its contribution to the right derivative is more elaborate, because in such case adding an infinitesimal positive amount, ε, breaches the next block (be it degenerate or not), so it should be appended to the block of r_j. If so, let l denote the index of the head activity of the next block, and then set $v_{sj}^{+} = v_{sj} + v_{sl}^{+}$. If we would actually increase r_j (as we might do during a search for the best r_j value), then we would also have to set $v_{sl} = 0$; that is, v_{sj} would inherit the contribution of v_{sl}. If $v_{sl}^{+} \neq v_{sl}$, then one or more downstream blocks will also have to be adjusted, iteratively, but we omit further details. Considering all S contributions, the right partial derivative is given by $V_j^{+} = \sum_{s=1}^{S} v_{sj}^{+}$. Calculating the left derivative is similar but now the question is not whether the block starting at r_j matches a downstream release date but whether an upstream block matches r_j. If so, then the contribution of the scenario to the left derivative by r_j becomes 0, and we write $v_{sj}^{-} = 0$; in such case, if we would actually reduce r_j during a search, its previous value would be inherited by the relevant upstream block. However, we do not need to consider chain reactions because the only new breaching occurs at r_j. In general, block structure changes at points where the regular derivative is not properly defined (which is the case if a scenario exists for which a block matches the next release date), but these points are especially relevant for our search. At all other points $v_{sj}^{+} = v_{sj} = v_{sj}^{-}$; and if that is the case for all scenarios, then the left- and right derivatives are the same (and we can write $V_j^{+} = V_j = V_j^{-}$). Note that $V_j = \sum_{s=1}^{S} v_{sj}$ is always well defined; it's just not always a legitimate derivative. Below we discuss the possibility of searching for the optimal solution by generic search algorithms that require the gradient as input, and if so then in practice we can use V_j as the partial derivative by r_j. The price is that we will not be able to match the precise optimal solution for the given sample.

We are now ready to show that, under independence and for converging networks, Algorithms RN19.1 and RN19.2 are equivalent, but not identical. Again, they are not identical because they respond to sampling errors differently. What we show is that under independence (and only under independence) the expected gradient after running Algorithm RN19.1 is zero and the solution is unique; we also show that Algorithm RN19.2 converges to the same solution and that's why we can say the two algorithms are equivalent. However, possible sampling errors imply that Algorithm RN19.2 may have more than one solution (unlike Algorithm RN19.1). Even if that happens, for a sufficiently large S, the difference should be negligible. In general, sampling errors are due to randomness but in our case there is an additional source of error because we use matching, which is never perfect (it would be perfect if the height of the vertical steps would be zero instead of $1/S$). Both sources are reduced by increasing S. In what follows, for convenience, we ignore matching errors; equivalently, we can say that we assume S is sufficiently large to render matching errors negligible.

We begin with the serial case and proceed by induction. For $k = 0$, Algorithm RN19.1 sets r_n to d minus the $\lceil S(1-\alpha_n) \rceil^{th}$ value in the set $\{p_{s,n}\}_{s=1,\ldots,S}$. Because we ignore matching errors, and while we also ignore upstream breaches, we can say that $S\alpha_n$ scenarios are critical. Therefore, this selection satisfies Step 2 of the algorithm *and* it is the only solution that matches the derivative by r_n to zero (under the assumption that it is not dominated by any upstream release date). For $k = 1$ (while setting r_{n-1}), let f denote the fraction of scenarios that breach r_n; a fraction with an expected value of α_n of the Sf breaching scenarios already breached the due date at the previous stage. The expected value of the latter fraction is precisely α_n due to independence (and it would be different under dependence). So, before accounting for new critical scenarios, the Sf breached scenarios contribute an expected value of zero to the derivative by r_{n-1}; and, by the larger sample validity principle, they do not change the expected value of the derivative by r_n, leaving it balanced. In addition, all S scenarios together contribute $-S\alpha_{n-1}$

to the derivative by r_{n-1}. To match that to zero, $S\alpha_{n-1}$ scenarios that were not critical in any previous stage must now become critical. But that's precisely what Algorithm RN19.1 requires. Note further that if we tighten r_{n-1} any more, the number of new critical scenarios can only grow, so the solution is unique. The same argument applies for $k \geq 2$ with breaching by activity $n-k$ applying at all levels downstream, namely $n-k+1, n-k+2, \dots, n, n+1$. The main observation is that the expected change of the derivative by breaching is zero both at the $n-k$ level and at the lower levels. Hence what remains is to balance $-S\alpha_{n-k}$ against a proportion of α_{n-k} of new critical scenarios (that were not critical at any lower level) and this implies the expected value of the derivative is zero. Again, at each stage, only one solution exists because breaches do not change the expected derivative and thus the number of new critical scenarios at each stage is predetermined and only one solution satisfies it. Hence, there is only one release date that satisfies the derivative by r_n, and, by induction, only one set of unique release dates with zero gradient (assuming that if a release date is dominated during the process we do not change it, so it remains unique). But we need independence for this purpose because without independence we cannot invoke the larger sample validity principle. If so, we cannot assume that breaches do not change the expected value of the derivative and therefore we cannot rely on the count of new critical scenarios as an indication that the derivative is matched to zero. Indeed, because Algorithm RN19.2 does not rely on these assumptions, and because sampling errors imply that breaching changes the derivative, it can happen that the zero gradient solution will be different and, possibly, not unique. Nonetheless, it is a sampling based solution of a model that has a unique solution in the neighborhood, and that implies that multiple solutions, if any, can only be due to sampling errors whose implications become progressively negligible when S grows large.

Now consider a converging network under independence and consider a version of Algorithm RN19.1 that allows iterations. Balancing requires such iterations but when it's done, all inherited blocks still contribute zero to the expected derivative value and the essential balance required to match the derivative to zero is the same as in the serial special case. In fact, iterations are required for one purpose only: to make sure that the correct number of new critical scenarios is achieved upstream of assemblies, because without iterations some new critical scenarios previously created may be taken over by a parallel activity (as discussed above). Hence, Algorithm RN19.1 still has one unique solution with a zero-expectation gradient and Algorithm RN19.2 will converge to the same unique solution.

One difference between PAP and PAR models is that under PAR a zero gradient is not sufficient for optimality. Even in the independent serial case, unless activity 1 dominates all others, there are two zero-gradient solutions. That is because if we set all release dates to the same value but such that the criticality is β, then every scenario has a single block starting at r_1. The partial derivatives downstream of r_1 are zero because their blocks start upstream, at r_1. The derivative by r_1 is zero because the criticality is set to β. But in typical cases where at least one other activity is not dominated by activity 1, it can be shown that this solution is not optimal. Notably, both Algorithms RN19.1 and RN19.2 avoid this suboptimal solution by proceeding backwards. If activity 1 does dominate all others, both algorithms will set r_1 no earlier than the other release dates and thus they will identify the optimal solution after all.

A Special Serial Case: Ordered Activity Durations

In sample-based analysis, if $p_{sj} \leq p_{tj}; \forall 1 \leq s < t \leq S, 1 \leq j \leq n$, then the scenarios are *ordered*. When scenarios are ordered, the scenario with the k-longest jth activity among all scenarios also has the k-longest first activity, second activity, etc. One way to generate a sample with ordered scenarios is to make all activities in each scenario perfectly dependent on each other, with a correlation coefficient of 1, and then sort the scenarios from smallest to largest. Another way to generate such a sample is to take any existing sample, sort all the activities one by one, and re-index the scenarios from smallest to largest for each activity. We show that if a serial project has ordered activity durations, it can be solved analytically not only for PAR but also for PAP. That is, no iterations are required. We start by solving the serial ordered case under PAP by a single-pass algorithm, and then we show how to accommodate PAR by the same algorithm.

Whereas we know that the PAP model is convex for any distribution, and thus has a unique optimal solution with zero gradient, recall that in general we require iterations to find that solution. Emphatically, this applies to the serial case as well: there is no known general one-pass algorithm for the serial PAP case. To present the one-pass, analytical solution for the ordered serial case, we first assume continuous distributions. If $F(x) = y$ is the cumulative distribution function (cdf) of a continuous duration, then $F^{-1}(y) = x$; in other words, F's inverse is well defined. The key to the PAP solution of the ordered serial case is that we know the desired criticalities of all release dates in advance: if we denote the desired criticality of r_j by w_j then $w_j = \alpha_j; j = 1, \ldots, n$ (regardless of whether durations are ordered or not). Furthermore, it turns out that it is sufficient to breach or match the next release date to achieve criticality. Let $F_k(p)$ be the distribution of p_k, the duration of activity k, and let $F_k^{-1}(w)$ be its inverse. Let $W_k = \sum_{j=1}^{k} w_j$; in particular, under PAP, $W_1 = \alpha_1$, and $W_n = \sum_{j=1}^{n} \alpha_j = \alpha$ (because $w_j = \alpha_j$). We also define $W_0 = 0$. To find the optimal release dates, we schedule backwards starting with r_n, so when we schedule r_j, we already know r_{j+1}. As we show presently, the optimal r_j is then given by $r_j = r_{j+1} - F_j^{-1}(1 - W_j)$. The sample-based version of this solution is given by Algorithm RN19.3.

Algorithm RN19.3

Setting Release Dates for Ordered Serial Projects under PAP

Step 1. Initially, let $k = 0$ and let $r_j = 0$ for $j = 1, \ldots, n$.

Step 2. Ignoring all upstream activities, postpone (that is, increase) r_{n-k} incrementally (thus tightening $r_{n-k+1} - r_{n-k}$) just enough so that at least a total of $\lceil SW_{n-k} \rceil$ $p_{s,n-k}$ values match or breach r_{n-k+1}. With a sorted sample that is achieved for $s = \lceil S(1 - W_{n-k}) \rceil$.

Step 3. If $k < n - 1$, increment k by 1 and return to Step 2. Else, stop.

To see that Algorithm RN19.3 yields the optimal solution, and likewise in the continuous case above, first note that r_n is set with an initial criticality of $\alpha = \sum_{j=1}^{n} \alpha_j$. But in the next step, when we set r_{n-1}, all but α_n of that criticality is breached (and thus shifted upstream) by r_{n-1}, whereas no new scenario becomes critical. Hence the remaining criticality of r_n is α_n, as required. Similarly, the next step leaves a net criticality of α_{n-1} still controlled by r_{n-1}, and so on. That is true because the sample is ordered so the scenarios whose tardiness is increased during the procedure are always a subset of the ones whose tardiness increased in the previous step. That is, the first step renders the longest $SW_n = S\alpha$ critical; the second step increases the tardiness of the longest $SW_{n-1} = S(\alpha - \alpha_n)$, leaving the tardiness of $S\alpha_n$ controlled by r_n, and so on. The end result is that r_j controls a criticality of α_j (for $j = 1, \dots, n$), as required by the PAP model. A particularly convenient feature of the algorithm is that at each step we deal with a single activity: breaching the next due date is sufficient to guaranty the block thus created breaches the due date.

Fortuitously, it is also possible to address the ordered serial case under PAR by the same solution, because there is a way to find the optimal criticalities of all release dates and use them as input for Algorithm RN19.3 or for its continuous version, as necessary. Furthermore, these optimal criticalities are strictly positive, so the structure of the solution of PAR is similar to that of PAP: it is guaranteed that there will be no passive release dates. The key here is the block structure of the optimal solution (in both cases). Because in each step we increase the tardiness of a subset of already tardy scenarios—as discussed earlier—all blocks are either degenerate or critical. By contrast, in the general case, non-degenerate blocks may terminate upstream of activity *n*, and not all blocks that include activity *n* are necessarily critical. In particular, even in the ordered case, if we allow converging networks then some blocks will terminate just above each assembly. But the observation holds in the serial case, and we can use it to calculate the desired criticalities, w_j, that drive the partial derivatives by r_j to zero under PAR. Starting with w_1 (which equals W_1), we can write

$$W_1 = w_1 = \alpha_1 + w_1 \sum_{j=2}^{n} \alpha_j$$

That is true because by definition, a fraction of all scenarios, w_1, is critical and it pushes the release date backwards by a weight proportional to w_1, whereas in the opposite direction we have α_1 plus the total weight of the activities that follow activity 1 in the critical block. The latter is given by the rightmost product. Balance implies that these two forces should be equal. Equivalently,

$$w_1 = \frac{\alpha_1}{1 - \sum_{j=2}^{n} \alpha_j} = \frac{\alpha_1}{\beta + \alpha_1}$$

At the r_2 level, a proportion of W_1 is breached from above, leaving $1 - W_1$. Therefore, similar analysis to the previous case now leads to

$$w_2 = \frac{\alpha_2(1 - W_1)}{1 - \sum_{j=3}^{n} \alpha_j} = \frac{\alpha_2(1 - W_1)}{\beta + \alpha_1 + \alpha_2}$$

And in general,

$$w_k = \frac{\alpha_k(1 - W_{k-1})}{1 - \sum_{j=k+1}^{n} \alpha_j} = \frac{\alpha_k(1 - W_{k-1})}{\beta + \sum_{j=1}^{k} \alpha_j}$$

By adding up the expressions for w_1 and w_2 we obtain

$$W_2 = w_1 + w_2 = \frac{\alpha_1 + \alpha_2}{1 - \sum_{j=3}^{n} \alpha_j} = \frac{\alpha_1 + \alpha_2}{\beta + \alpha_1 + \alpha_2}$$

which implies that, for the purpose of calculating w_j for $j \geq 3$, the first two activities can be combined to a single representative activity with a total weight of $\alpha_1 + \alpha_2$. Hence, by induction

$$W_k = \sum_{j=1}^{k} w_j = \frac{\sum_{j=1}^{k} \alpha_j}{1 - \sum_{j=k+1}^{n} \alpha_j} = \frac{\sum_{j=1}^{k} \alpha_j}{\beta + \sum_{j=1}^{k} \alpha_j}$$

By setting k to n, and noting that $\sum_{j=n+1}^{n} \alpha_j = 0$ or that $\beta + \sum_{j=1}^{n} \alpha_j = \beta + \alpha = 1$, we now see that $W_n = \sum_{j=1}^{n} w_j = \sum_{j=1}^{n} \alpha_j = \alpha$, as might have been expected. Furthermore, there is no need to calculate all the w_j values explicitly. That is true because the input we need is W_k, not w_k. Nonetheless, the W_k expression makes it possible to calculate w_k for any desired $k \geq 2$ without calculating any previous values by

$$w_k = W_k - W_{k-1} = \sum_{j=1}^{k} w_j - \sum_{j=1}^{k-1} w_j$$

$$= \frac{\sum_{j=1}^{k} \alpha_j}{1 - \sum_{j=k+1}^{n} \alpha_j} - \frac{\sum_{j=1}^{k-1} \alpha_j}{1 - \sum_{j=k}^{n} \alpha_j}$$

$$= \frac{\alpha_k (1 - \alpha)}{\left(1 - \sum_{j=k+1}^{n} \alpha_j\right)\left(1 - \sum_{j=k}^{n} \alpha_j\right)}$$

$$= \frac{\alpha_k \beta}{\left(\beta + \sum_{j=1}^{k-1} \alpha_j\right)\left(\beta + \sum_{j=1}^{k} \alpha_j\right)}$$

In particular, for $n \geq 2$, $w_n = \alpha_n (1 - \alpha)/(1 - \alpha_n) < \alpha_n$ but $w_1 = \alpha_1/(\beta + \alpha_1) > \alpha_1$ (because $\beta + \alpha_1 < 1$) . Furthermore, all w_j values are strictly positive, which is why the solution does not involve dominance: all *n* release dates are active. Example RN19.4 is based on Example RN19.1, but with ordered durations.

Example RN19.4. Consider a serial project with 3 PAR activities and a due date of $d = 200$. The durations of the activities are ordered and each follows a lognormal distribution with *cv* = 1 and means $\mu_1 = 50$, $\mu_2 = 10$, and $\mu_3 = 100$. Earliness costs (α_j) per scenario are 0.07, 0.14, and 0.49, respectively. Hence the tardiness cost per scenario is $\beta = 0.3$. Find the optimal W_j and r_j values.

Calculating the required W_j values, we get

$$W_1 = w_1 = \frac{\alpha_1}{\beta + \alpha_1} = \frac{0.07}{0.3 + 0.07} = 0.189189$$

$$W_2 = \frac{\alpha_1 + \alpha_2}{\beta + \alpha_1 + \alpha_2} = \frac{0.07 + 0.14}{0.3 + 0.07 + 0.14} = 0.411765$$

and $W_3 = \alpha = 0.7$. As expected, $w_1 = 0.189189 > 0.07$, and $w_3 = 0.288235 < 0.49$ (whereas $w_2 = 0.222576$). One way to find the optimal release dates for this project is to use a standard normal table or a spreadsheet normal inverse function. That yields $r_3 = 154.3043$, $r_2 = 145.7906$, and $r_1 = 72.1765$. Another way is to use the same sample of 1001 scenarios that we used before, but with sorting. By this choice, some of the sampling errors are unchanged. By implementing Algorithm RN19.3, we then obtain $r_3 = 152.0504$ (compare to 154.3043, the original value by Algorithm RN19.1, 126.8421, and its final bounded value of 134.2488), $r_2 = 143.6972$ (compare to 145.7906 and to the original value 134.2488), and $r_1 = 63.5379$ (compare to 72.1765 and to the original value of 69.8741). The objective function value for this example is 102,933.2 (compare to the original value of 106,980.8). To get an idea about the difference in total cost between the inverse-normal-generated release dates and the ones based on Algorithm RN19.3, we inserted the calculated release dates into the sample-based calculation. The result was an increase of 0.05%. Note that this increase is not due to any inferiority of the calculated release dates, in fact it is the reverse! But the sample-based release dates are optimal for the given sample. Hence, this result suggests that using the sample here involves a sampling error that is likely to cause a small fraction of a percent increase in the objective function.

The most important difference between Algorithms RN19.1—which is valid under independence—and Algorithm RN19.3—which is valid for ordered samples—is that the latter can never yield dominated release dates. This suggests that dominance is less likely when durations are positively correlated than when they are independent (because the ordered case is fully positively correlated). In particular, we might expect that dominance will be less likely under linear association than under independence. Recall from Chapter 18 that positive correlation is likely in practice and that it can be modeled by linear association.

An Engineering Approach for General PAR Networks

To our knowledge, the PAR problem has only been formally solved for converging networks, which are quite simple project structures, and subject to independence, which we know to be invalid in practice (see Chapter 18). But at the very least we can provide satisfactory sample-based engineering solutions for any network structure, and independence is not necessary. We can do that simply by matching the gradient of the objective function to zero. (The question whether more than one local solution can exist for various models beyond converging networks under independence is open.) Furthermore, in practice it is enough to drive the gradient to a sufficiently low value instead of insisting on perfect matching. Our aim in this section is to describe how a generic search program can solve PAR for any network structure. In the next section, we generalize further to include mixed PAR and PAP activities. Until further notice, we rely on generic search programs and limit derivative calculations to V_j. Later, we outline the option of solving the same models by a dedicated program that matches the gradient to zero precisely, as in Algorithm RN19.2, but that requires distinguishing between left- and right derivatives explicitly (so V_j is not sufficient).

From an engineering point of view, small samples are risky. For instance, in Example 19.1, which is a small ACM instance with n = 10, using a sample of 100 instead of 10000 would lead to suboptimality of 0.9% whereas a sample of 1000 reduces the optimality gap to 0.2% (p. 545). The upshot is that sample size matters, and we must use reasonably large samples. But on the other hand, if we use a sufficiently large sample then it is not very important to match partial derivatives—that is, the gradient—to zero precisely. In other words, it is not imperative to distinguish between left- and right derivatives and only select release dates with undefined derivatives (and such that only the left derivative is negative). Although the scaling we chose gives each scenario a weight of unity, the true weight of a scenario is proportional to 1/S, and we can make that as small as we wish by increasing S. In practice, at least from a programming point of view, it is more efficient to use a generic search routine that relies on knowing the gradient. We do not recommend using a generic search program that does not utilize derivatives because in our case it is easy to identify blocks while calculating the objective function value and therefore calculating V_j is relatively easy. In more detail, above, we showed the connection between blocks and partial derivatives. Although we limited our analysis to converging networks, it is clear that the only difference between those special cases and general networks is that blocks may not be serial (we provide examples below). The key observation is that the release date of a block controls all the activities in the block, and that does not require the block to be serial. Hence it is straightforward to calculate measures such as v_{sj} for blocks of any structure, and they still only require very simple programming (akin to CPM calculations of the type covered in Chapter 16 for deterministic projects). In this section we propose using a generic search program rather than building a more complex program that extends the capabilities of Algorithm RN19.2. Assuming the program requires gradient values, we can use $V_j = \sum_{s=1}^{S} v_{sj}$ as the partial derivative by r_j (subject to mild conditions that we discuss later). Recall that V_j is always well defined. For this search, it may be beneficial to start with a small S and then use the solution as a hot start with progressively larger samples. In such case, we can stop increasing S when the solution converges to a stable objective function value. If S is sufficiently large and the objective function value has stabilized, there is no compelling practical need to match the gradient to zero perfectly.

Figure RN19.2 is a reproduction of Figure 19.2, showing the interdictive graph with release dates in AOA representation, but we use activity names instead of node pairs; for instance, A instead of 2-4 (following the original text notation, however, r_A does not denote an assembly activity here). Recall from Chapter 16 that when a project network embeds the interdictive graph it cannot be decomposed, and thus it resists probabilistic analysis. So in a sense the interdictive graph is the most basic project network that is difficult to analyze. It is certainly more complex than a series-parallel network let alone a converging network. Nonetheless, in Chapter 19 we showed how to find PAP release dates for it. We use the figure to show how our definition of blocks applies to projects whose network includes this complex structure element. And if we can identify blocks, we can also calculate derivatives. In other words, at least locally, we can find optimal release dates for the interdictive graph—and thus really for any project network—not only for PAP but also for PAR.

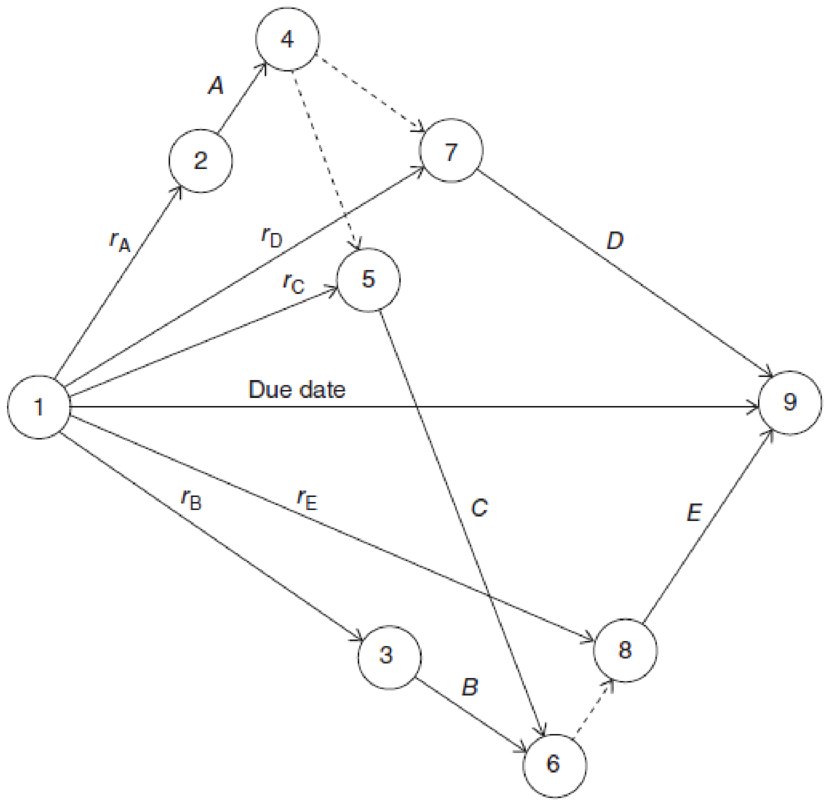

Figure RN19.2: *The Interdictive Graph with Release Dates (AOA)*

Example RN19.5. Consider the project depicted in Figure RN19.2. Let $\alpha_j = 0.05$ for all activities, set $r_A = 1$, $r_B = 2$, $r_C = 2$, $r_D = 3$, $r_E = 3$, and $d = 4$. Identify the block structure and provide derivative calculations for a scenario, s, where each activity takes precisely 1 time unit. Would it make a difference if all activities were shorter?

In this scenario all activities start precisely at their respective release dates and the project is weakly critical. Because there is no breaching anywhere, we obtain five degenerate blocks each associated with one of the release dates. Therefore $v_{sj} = -0.05; j = \mathrm{A, B, C, D, E}$. Regarding the contributions to the left derivatives, A and B are root activities so $v_{sj}^{-} = v_{sj} = -0.05; j = \mathrm{A, B}$; each of the remaining activities has at least one upstream block abutting its release date, and hence $v_{sj}^{-} = 0; j = \mathrm{C, D, E}$. To calculate the contributions to the right derivatives, it is convenient to work backwards. Starting with E, adding an infinitesimal positive amount, ε, to the duration, the due date will be breached by the block starting at E. So the scenario's contribution to the right derivative of E is $v_{s\mathrm{E}}^{+} = 1 - 0.05 = 0.95$. Using T to denote strict tardiness, the block involved is E-T. The same calculation applies for D, and the block would be D-T. When we consider C, on the right it will breach E and create a tardy block, C-E-T, with a right derivative contribution of $v_{s\mathrm{C}}^{+} = 1 - 0.1 = 0.9$. Again, the same calculation applies for B, with the block structure B-E-T. Delaying A leads to the block A-C-E-T, with $v_{s\text{A}}^{+} = 1 - 0.15 = 0.85$. Finally, if all activities would be shorter, the project would not be weakly critical and no non-degenerate blocks would be created when calculating left- and right derivatives; hence we would obtain $v_{sj}^{-} = v_{sj} = v_{sj}^{+} = -0.05; \forall j$.

Example RN19.6. Consider the same project as in Example RN19.5, but now assume the scenario has activity times of precisely $1 + \varepsilon$ time units each, where $\varepsilon \ll 1/3$.

In this scenario, all activities exceed their allowances by ε. In particular, A, B (the root activities) and D start on time. r_C is breached by ε and therefore C breaches E by 2ε, leading to strict tardiness of 3ε; the block involved is A-C-E-T. By itself, B would delay E by ε, but because C breaches it by 2ε, B remains a degenerate block. Similarly, D would delay the project by ε, but the A-C-E-T block delays it by 3ε, so D too is a degenerate block. In summary, the block structure is A-C-E-T, B, and D. Because the due date is breached by it, the contribution of A-C-E-T to the derivative by A is $v_{s\text{A}} = 1 - 0.15 = 0.85$; B and D, as degenerate blocks, retain their levels of $v_{sj} = -0.05; j = \text{B}, \text{D}$; C and E now have zero contributions, having yielded them to A, so $v_{sj} = 0; j = \text{C}, \text{E}$. Finally, $v_{sj}^{-} = v_{sj} = v_{sj}^{+}; \forall j$.

Example RN19.7. Consider the same project as in Example RN19.5, with durations of 1 each, but now add a preliminary preparation activity connecting a new node (indexed 0) to node 1, with a release date of 0 and the same weight (0.05); assume all release dates now start at node 0 (instead of node 1). Identify the block structure and provide derivative calculations for a scenario where each original activity takes precisely 1 time unit but the preliminary activity takes 1.5 units. Repeat for a scenario where it takes 2.5 units.

First, note that A and B are no longer root activities. If the new activity takes 1.5 time units, then it breaches A by 0.5, which in turn breaches C and E. But B and D are not breached. If we denote the new activity by P, the block structure is P-A-C-E-T, B, and D. B and D retain their contributions of -0.05 each, A yields its contribution to P, and because the due date is breached, P's contribution is $1 - 0.2 = 0.8$. There is no difference between the right and left derivatives of the block starting at P, and because B and D would not breach the due date even if they would take slightly longer, their left and right derivatives are the same too. But if the duration of P grows to 2.5, then B and D are breached too, and the project is even tardier. As a result there is now a single block, including all six activities. This block is not serial: it has two divergences (A from B and C from D) and two assemblies (B with C and D with E). The derivative contribution—all taken by P—is $1 - 0.3 = 0.7$. As in the previous example, the contribution of all other activities are 0 and $v_{sj}^{-} = v_{sj} = v_{sj}^{+}; \forall j$. To check whether the result is correct, note that if we would postpone (advance) r_P from 0 to some positive (negative) value, all activities would be delayed (advanced) by the same amount, leading to a gain (loss) of 0.3 per unit in holding costs before the due date, but also to a tardiness penalty (gain) of 1 per unit. Considering the assembly points, both B and D would have to wait for their pairings at those assemblies, but they still belong to P's block. (However, a dynamic approach might allow postponing their release dates to match the known delay of P with appropriate safety, in which case they would likely become degenerate blocks after all.)

As these examples demonstrate, figuring out block structure for any scenario and calculating its contribution to derivatives is possible not only for serial projects and other special cases, but for any project network. For a given scenario, s, and a given release date, r_j, we have to identify which activities are delayed when the release date is tightened and which start early when it is loosened. Essentially, for the purpose of calculating the right-derivative of r_j and for scenario s, our definition of a block implies that the set of all activities that are delayed when r_j is tightened are indeed a block, and for the left-derivative, the block is the set of all activities that start earlier if r_j is loosened. Left derivative contributions are different from v_{sj} only if an upstream block abuts r_j, in which case the left derivative contribution becomes zero. In the case of divergent activities, a block may then have a tree structure, and as our examples demonstrate, blocks can even have multiple paths that diverge from each other and then remerge. However, we can also view each block as a diverging tree, regardless of the network structure behind it. Forking is likely, because two or more activities may await the same event, but it is highly unlikely that when paths that are controlled by the same release date converge, they do so at the same instant. For instance, consider Example RN19.7 with P = 2.5, where we saw that the whole project is in the block, but we could also say that the block has a tree structure because B and D finish without delaying any subsequent activity and thus they may be viewed as branches ending with leafs; that is, the stem of the tree is the series P-A-C-E-T, whereas B emanates from P's finish node, thus creating a branch, and D emanates from A's finish node, and also creates a branch. Figure RN19.3 depicts this block as a directed tree.

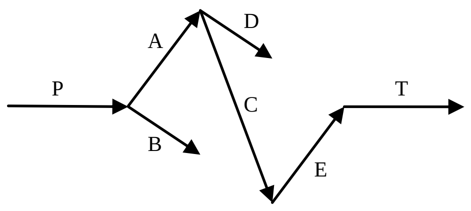

Figure RN19.3: *Tree Block Structure in Example RN19.7 (P = 2.5)*

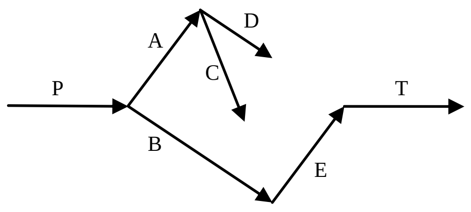

Figure RN19.4: *Tree Block Structure in Example RN19.8 (P = 2.5, B = 2)*

Essentially, an activity is appended to a block when it is breached by another activity that is already in the block, and its membership in the block does not depend on whether it, in turn, breaches another activity or not. So the structure of a block is really not important: only membership matters. However, although convergence is unlikely within a block—as we observed earlier—it may occur when a release date is adjusted just enough to match the start time of an activity that belongs a block that starts at another release date. Any further increase of said release date usurps part of the existing block. In other words, changing a release date affects not only the composition of its own block but that of other blocks as well: clearly any activity appended to a block must be yielded by another block. At the adjustment for which the release date just matches the start of an activity from another block, however, there is no new breaching yet, so neither block needs to be changed. Example RN19.8 illustrates one way that blocks can change.

Example RN19.8. Consider the same project as in Example RN19.7, where the preliminary preparation activity taking either 1.5 or 2.5 time units, but now assume that B takes 2 time units (and the other activities are not changed).

Recall from Example RN19.7 that if P = 1.5 then B starts at its release date, 2 (and thus does not belong to P's block). But under this new scenario B finishes at 2 + 2 = 4, whereas C finishes at 1.5 + 1 + 1 = 3.5 (< 4). Hence it is now B that breaches E, and creates a new tardy block, B-E-T. The P block is not tardy in this scenario and is given by P-A-C. And D is still a degenerate block. But if P = 2.5, the same full block we identified in Example RN19.7 remains valid, although the block's tree structure changes to P-B-E-T as the stem, with A as a major branch from P, splitting further to C and D (see Figure RN19.4). Notably, when P = 2.5, the partial derivative by r_P does not change relative to Example RN19.7 because the block membership does not change (even though its structure does), but if P = 1.5 and B now takes longer, block memberships change quite dramatically.

Example RN19.9. Consider the project depicted in Figure RN19.2, but now assume that two preliminary activities in series precede A and B, denoted as O and P. The holding costs per scenario are $\alpha_O = 0, \alpha_P = 1/12,\ \alpha_A = 1/6,\ \alpha_B = 1/12,\ \alpha_C = 1/4,\ \alpha_C = 1/4,\ \alpha_D = 1/12,$ $\alpha_E = 1/12$, and $d = 40$. Ignoring r_P, the other release dates are $r_O = 0$, $r_A = 10$, $r_B = 21$, $r_C = 22$, $r_D = 31$, $r_E = 32$, and $d = 40$. In a particular scenario, Activities O, P, A, B, C, D, and E take 3, 5, 8, 5, 7, 5, and 6, respectively. For this scenario, show the derivative contribution by r_P as a function of r_P.

If $r_P < 3$, then P is part of the block starting at O, and therefore the contribution to the partial derivative by r_P is zero between 0 and 3. At that threshold, the contribution drops by $-\alpha_P = 1/12$, and stays at that level until r_A is breached, at which point it drops further by 1/6; that takes place when $r_P + 5 > r_A = 10$, just to the right of $r_P = 5$. Once A becomes part of the block, it completes at $r_P + 5 + 8 = r_P + 13$, and it breaches C at $r_P + 13 = r_C = 22$, leading to $r_P = 9$; just to the right of that point the contribution drops by 1/4 to $-1/2$. Once in the block, C follows A directly and takes 7 units, so it completes at $r_P + 20$; therefore it breaches E at $r_P = 12$, at which point the contribution drops to $-7/12$. Once in the block, E completes at $r_P + 26$, so the due date (40) is breached just to the right of $r_P = 14$, where the contribution increases by 1 to

5/12. B only joins the block to the right of $r_P = 16$, because $r_B = 21 = 16 + 5$, and at that point the contribution drops from $5/12$ by $\alpha_B = 1/12$ to 1/3. Finally, D joins the block at $r_P = 18$, where the contribution drops to 1/4. Figure RN19.5 shows these results. Recall that for $r_P = 14$, the contribution increases from $-7/12$ by 1 to $+5/12$. In the figure, these levels are marked as $\geq -\alpha$ and $\geq \beta$ because in general the contribution cannot fall below $-\alpha$ where it is negative, and cannot fall below β where it is positive. In our example it drops from $5/12$ to $1/4 = \beta$ after B and D are appended to the block. In general, there is a clear globally minimal *negative* value, illustrated in the figure just before the due date is breached, but the function is not necessarily unimodal because it may decrease in the positive range. As a result, there is no guarantee that local optima are also global optima. In more detail, the optimal solution for an individual scenario is clearly at the point where the derivative contribution of that scenario changes sign, or at least becomes nonnegative (14 in the example), but when we consider the sum of S contributions there is no longer a guaranteed unique argument at which this must occur: there may be more than one local minimum. We note, however, that in converging networks partial derivatives do not change in the positive side (although *other* release dates may change them).

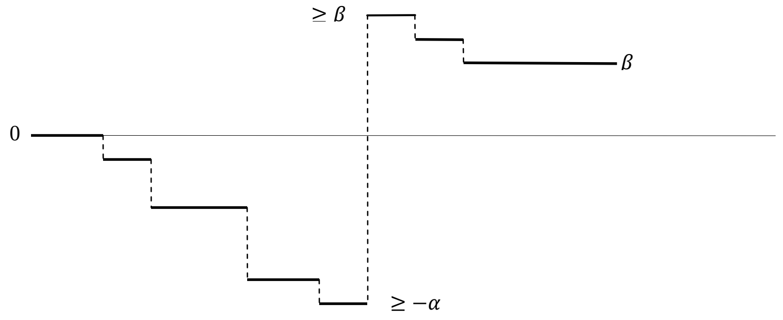

Figure RN19.5: *The Solution of Example RN19.9*

The calculations involved in such examples are straightforward and easy to program. Therefore, we can use a generic search program for the solution, and it can be one that requires the gradient. Again, for this purpose we propose using V_j as the partial derivative by r_j. If processing times are continuous and given with high precision, then it is not likely that any set of release dates generated by a generic search program (conducting a fine search, with practically continuous search values) will hit a point where any derivative is undefined or where more than one scenario changes at the same instant; that is advantageous and justifies the use of V_j as the partial derivative by r_j, as we propose. Accordingly, for generic search purposes it is indeed recommended to use full precision when generating samples (even though it is not inherently important). Rounding of the resulting release date decisions, if desired, may be applied to the final search results. But if the simulation involves the Parkinson distribution, then problems such as undefined derivatives become possible again. Adding or subtracting a very small random noise value to Parkinsonian durations (before running the search) can resolve that problem without substantially changing the validity of the sample. If so, hitting discontinuities becomes virtually impossible and we can simply ignore the distinction between left- and right derivatives (which the use of V_j implies). In addition, typically, when running a search program that relies on gradient calculations, a legitimate stopping criterion is a reduction of the absolute values of all n

partial derivatives to below some sufficiently low threshold. However, in our case, it is simpler to use convergence to a stable objective function value as a stopping criterion. If we insist on using the partial derivatives, such programs typically allow the user to specify that threshold, and therein lies the complication: the partial derivative threshold value, t, must not be specified below 1/2 (although that is not restrictive because compared to a sufficiently large S, $t = 1/2$ is sufficiently low). A lower threshold than 1/2 is forbidden because it can prevent the program from converging. To see that, recall that when the due date is breached by any scenario, a penalty of 1 is charged for that scenario and subsequently added to the derivative—which is the sum of S scenario contributions. In the other direction, when tardiness is reversed, the scenario contribution to the derivative is reduced by 1. Therefore, if $t < 1/2$, then it can happen during the search that $-0.5 \leq V_j < -t$ (or, symmetrically, $0.5 \geq V_j > t$), in which case if we increase (decrease) the derivative by 1 the result will be $|V_t| \geq 0.5 > t$, and hence the derivative will straddle the range $[-t, t]$ and the algorithm will never converge. If we allow the use of rounded processing times, it may become impossible to converge even with higher threshold values. However, these issues relate to the use of a generic search program. Otherwise, a dedicated program could resolve the same issues in many different ways. For example, we could specify that if a small change in the search value causes a change of sign in the derivative, then the search can stop with that search value as the solution. Thus, such a dedicated program could also handle highly rounded sample values or be programmed to use nonparametric bootstrap resampling (Subsection 18.7.3). In Chapter 19, although we only addressed PAP, Example 19.1 relied on rounded durations and assumed the use of a dedicated program, whereas Example 19.3 rounded the final release dates instead. Both approaches have advantages and disadvantages. We suspect the same applies for PAR as well, and later we describe a possible dedicated program that uses rounded data. As in Example 19.1, a dedicated program can actually benefit from rounding.

An Engineering Approach for Mixed PAR and PAP

As we discussed already, the PAP model is appropriate for large projects, but subprojects may allow using PAR, and thus reduce costs. In practice, however, even a large project may have some activities that justify the use of PAR. In principle, a mixed model that allows both cost types is advantageous. Even the same activity can have two release dates for PAP and PAR aspects; for instance, we may have to order bespoke materials in advance but off-the-shelf materials and some workers can be sourced just in time. If so, and even if the PAR release date coincides with the PAP one, the PAR cost will not be incurred until the PAP activity actually starts. In such case, formally, a start-start relationship is called for, ensuring that the PAR parts will not precede the PAP expenses (since that would be useless), but that can be assured by a dummy activity of zero duration inserted after the PAP release date and before the PAR one. Holding costs are then allocated to the two release dates based on their actual magnitudes. (Under PAP, such aspects would be considered part of the same activity, with a combined holding cost.) Almost needless to say, a solution that allows both types of release dates can handle this case as well. Fortunately, it is not too difficult to generalize the PAR solution above, so we can address mixed models.

Whereas PAP-, PAR- and mixed models yield different optimal results, the key to the solution is that, for any scenario, block structure is determined by release dates only, not by the expenses involved. With this in mind, the only difference between PAP and PAR release dates is in how we calculate partial derivatives. Optimal solutions are different only because the gradient is different, and it in turn depends on whether release dates in a block are PAP or PAR. In that context, recall that root release dates are always PAP, but in the PAR model they still inherit criticality from downstream release dates that they breach (that is, that are in their block) whereas under PAP, they don't. So the question is not which type of activity heads the block, but which types of activities are downstream. The following rule addresses the calculation of V_j and has two parts:

1. Once a block is identified starting at r_j, when calculating v_{sj}, ignore any downstream PAP activities in the block: only activity j itself and breached PAR activities contribute negative weights to the derivative. If the strict tardiness pseudo-activity is in the block, it contributes +1.

2. When calculating V_j for a PAP release date, each scenario contributes $-\alpha_j$ even if r_j is breached. Specifically, if r_j is breached, then $v_{sj} = -\alpha_j$.

Together, the two parts imply that a breached PAP release date, r_j, does not contribute $-\alpha_j$ to the partial derivative of the upstream release date that controls the breach; instead, that contribution remains part of the derivative by r_j. In summary, only an un-breached release date can inherit contributions from downstream activities, as per Part 1. Part 1 does not distinguish between PAR and PAP activities. The logic behind Part 1 is that PAP activities do not yield criticality to upstream release dates because breaching them does not reduce their holding costs. Part 2 complements Part 1: PAP release dates do not yield criticality to breaching upstream release dates, if any, so they must retain it. But at the head of a block, there is no difference between a PAR and a PAP activity. That means that for a PAP release date, r_j, all S scenarios contribute $-\alpha_j$ to the partial derivative. In turn, that means we start with an initial value of $-S\alpha_j$. To that we add contributions from breached PAR activities in any non-degenerate block that starts at r_j, including tardiness penalties where appropriate. As a result, the criticality of a PAP release date is bounded from below by α_j. Another implication is that in an optimal solution, a PAP release date must have a proportion of at least α_j un-breached scenarios, because at least that many are required to eventually breach or match the due date (to drive the right partial derivative to zero). That is because a breached scenario cannot be critical: if the relevant block is critical, there must be an upstream release date that actually controls the criticality, and it is the one that earns the credit for it (that is, the penalty for it when tardiness is strict). For this purpose it does not matter if the upstream activity at the head of the block is PAP or PAR. A related observation concerns dominated PAR release dates (such as activity 3 in Example RN19.1). Being PAR, they start the partial derivative calculation process with zero (instead of $-S\alpha_j$ as in the PAP case), and because they are dominated they retain the optimal value of zero throughout the process. That is, unlike the PAP case, there is no compelling reason for a PAR release date to be active. Finally, V_j is sufficient for our purpose here because we rely on generic search, but if we focus on how blocks change in cases where either v_{sj}^- or v_{sj}^+ is not equal to v_{sj}, then it is straightforward to obtain the left- and right derivatives when it is important to match the gradient to zero precisely.

Example RN19.10. Revisit Examples RN19.5, RN19.6, and RN19.7 but now assume that C is a PAP activity. Update all block structures and derivative calculations.

Block structure does not depend on whether any activity is PAR or PAP, and hence the structures we identified before remain valid. The block structure in Example RN19.6 was A-C-E-T, B, and D. There is no change for B and D, but C is a member of the block A-C-E-T. Recall that we calculated a contribution to the derivative by A for this block as $1 - 0.15 = 0.85$, with 0 retained by C. But now C retains -0.05 and the contribution for A increases by 0.05 to $1 - 0.1 = 0.9$. The sum of these contributions is not changed, which is true for any scenario and given release dates. In Example RN19.7, for P = 1.5, the block structure was P-A-C-E-T, B, and D. B and D retain their contributions of -0.05 each because C is not involved with them, A still yields its contribution to P, but because C retains its own contribution of -0.05, P-A-C-E-T now has a contribution to P of $1 - 0.15 = 0.85$. Next consider Example RN19.8 with P = 1.5. The block structure was B-E-T, with a contribution to B of 0.9; P-A-C, previously with a contribution to P of -0.15 (when C was PAR), but now, for C as PAP, P's contribution is -0.1 and C retains -0.05; D remains degenerate with a contribution of -0.05. Now consider Examples RN19.6 and RN19.7 with P = 2.5, In Example RN19.8, the block structure was different from that of Example RN19.7 (see Figures RN19.3 and RN19.4), but membership was the same: all activities were in the same block and C was part of it in both instances. Hence the derivative contributions are the same in both examples, respectively -0.05 and 0.75 for C and P. Recall that for C as PAR, P had a contribution of $1 - 0.3 = 0.7$, and indeed $-0.05 + 0.75 = 0.7$.

Solving the Mixed Model for the Serial Ordered Case

For the serial ordered case, it is possible to calculate W_j for mixed models and implement Algorithm RN19.3. Retreating one step from the expression we had for w_k under PAR, we can write

$$w_k = \alpha_k(1 - W_{k-1}) + w_k \sum_{j=k+1}^{n} \alpha_j$$

Here the multiplication by $1 - W_{k-1}$ is required because, under PAR, balance requires that a fraction of α_k out of the *un-breached* scenarios be critical. By contrast, a PAP activity requires a fraction of α_k out of *all* scenarios. Hence if activity *k* is PAP, we should not include the multiplication by $1 - W_{k-1}$. Now consider the rightmost part of the equation. It accounts for inherited criticality from downstream breached PAR activities. Hence it should not include breached PAP activities (regardless of whether activity *k* itself is PAR or PAP). Denoting the set of PAR activities by *PAR*, if $k \in PAR$, we now have

$$w_k = \alpha_k(1 - W_{k-1}) + w_k \sum_{j>k, j \in PAR} \alpha_j$$

Whereas if activity *k* is a PAP activity, we have

$$w_k = \alpha_k + w_k \sum_{j>k, j\in PAR} \alpha_j$$

For $k = 1$, because $W_0 = 0$, the two expressions are equivalent, as they should be. Example RN19.11 is based on Example RN19.4, but with one PAP activity instead of PAR.

Example RN19.11. Consider a serial project with 3 input activities and a due date of $d = 200$. The durations of the activities are ordered and each follows a lognormal distribution with cv = 1 and means $\mu_1 = 50$, $\mu_2 = 10$, and $\mu_3 = 100$. Earliness costs (α_j) per scenario are 0.07, 0.14, and 0.49, respectively. Hence the tardiness cost per scenario is $\beta = 0.3$. Suppose that activity 2 is PAP and activity 3 is PAR. Find the optimal W_j values.

It is not necessary to specify the type of activity 1, since it always behaves as if it is PAP. Calculating the required W_j values, we get

$$W_1 = w_1 = 0.07 + w_1 \times 0.49 = \frac{0.07}{0.51} = 0.137255$$

$$w_2 = 0.14 + w_2 \times 0.49 = \frac{0.14}{0.51} = 0.274510$$

Leading to $W_2 = 0.411765$, or $1 - W_2 = 0.588235$. Plugging that value to the formula for w_3, which requires it as a PAR activity, we finally get

$$w_3 = 0.49 \times 0.588235 + w_3 \times 0 = 0.288235$$

which is unchanged relative to Example RN19.4, and still leads to $W_3 = 0.7 = \alpha$.

It is not straightforward to extend the serial ordered case to more general networks, because not all blocks are either critical or degenerate. But it is possible to use similar analysis to develop lower bounds on criticalities in more general cases. We omit further details, however.

Programming for a Precise Solution

In the previous sections, except for the serial ordered case, we assumed the use of a generic search program. To make it possible to use such a program we recommended high precision when generating samples and avoiding excessive sensitivity when specifying tolerance limits for the absolute value of any partial derivative. Both recommendations were motivated by the need to avoid numerical problems that may be associated with multiple paths being critical at the same time or with blocks abutting each other (which can be caused by rounded data). The price we have to pay for this approach is that we cannot identify the optimal solution precisely. That is because the optimal solution is always at a discontinuity of the type that our recommendations are designed to avoid. From an engineering point of view, insisting on finding precise solutions in balancing problems of the type we have here is redundant. Arguably, the imprecision associated with using sample-based optimization dominates the imprecision associated with not using the precise discontinuities that only apply to the specific sample we happen to use. Nonetheless, in this section we address the precise solution for general networks that allow a mixture of PAP and PAR release dates. Whereas we still allow the use of highly precise sample values, they are not required: we now allow rounding of the type the text introduced in Example 19.1. Almost needless to say, except for issues associated with stopping criteria of generic search programs, there is no fundamental difference between rounded and precise samples. And from an engineering point of view (but without considering stopping criteria), if rounding involves an imprecision that is much smaller than the standard deviation of the duration, it is indeed negligible. The reason we allow rounding here is twofold: first, the data we have may be rounded already; second, in a dedicated program we can program appropriate stopping criteria that can handle such complications. Hence, in what follows we assume rounding, and that is equivalent to assuming that all data and the solutions are integer (albeit typically large integers). In an optimal solution the release date should be the smallest possible integer for which $V_j^+ \geq 0$. For this purpose, calculating v_{sj} for $r_j - 1$ or for $r_j + 1$ is an effective way to obtain v_{sj}^- and v_{sj}^+, the contributions of Scenario s to V_j^- and V_j^+, respectively.

Conclusion

In Chapters 16 through 19—the project scheduling module of our text—we start with the basic established PERT/CPM theory and conclude with state-of-the-art models that improve on the traditional framework in several ways. Our main objective is to provide valid stochastic scheduling approaches that include safety time. To be valid, such approaches must be based on sound estimates of the relevant distributions. We adopt the lognormal distribution with linear association for this purpose, but we also recognize that the Parkinson effect may occur (with a lognormal distribution at its core). Here, we stress that the whole structure crumbles without a valid way to estimate distributions. That simple observation explains the failure of conventional PERT to adequately model stochastic scheduling. We propose PERT 21 as a more complete framework based on an improved stochastic engine. Although PERT 21 is a new framework, we make no claim that all of its ingredients are new. The basis is familiar as PERT, and many of the ingredients that are not always considered part of PERT have actually been known in the research community for a long time. In these cases, our message is that it is time to explicitly treat them as parts of one system. Even simulation—proposed over 60 years ago and widely recognized as the most practical approach to issues such as the Jensen gap—is not a standard part of basic project management software. There is also a need to remove such obstacles as the triplet method and the statistical independence assumption. The latter need was recognized by academics many years ago but not fully embraced in practice. Specifically, reliance on the triplet estimation method guides practitioners away from using historical data in support of PERT scheduling. This perspective may be appealing because projects are perceived as one-of-a-kind endeavors, so history is theoretically irrelevant. In fact, new projects have many factors in common with historical projects. Nonetheless, solid applications that involve reliable distributions based on history and safe scheduling models are almost nonexistent. We discuss a notable exception below.

In Chapter 16 we described and critiqued PERT/CPM. In Chapter 17 we covered the most important models for sequencing activities under deterministic assumptions. One of the effective ways to do so relies on adjacent pairwise interchanges (API) and related approaches such as tabu search, genetic algorithms, etc. It turns out that if we use the model of Trietsch et al. (2010) to estimate distributions, we obtain distributions that are stochastically ordered. The reason is that we can assume a consistent coefficient of variation (*cv*), and for the lognormal distribution, the cdfs of processing times with the same *cv* do not intersect each other. In such cases, we may be able to calculate the effects of API for stochastic cases without resorting to simulation. Our discussion of emerging research areas in our Research Notes for Chapter 6 can shed more light on this point. We refer to this approach as *lognormal scheduling*. Stochastic order also applies for the Parkinson distribution if its core is modeled by a lognormal with consistent *cv*, but adapting the calculations provided in our Research Notes for Chapter 6 for the Parkinson distribution requires further research.

As we showed in the chapter, it is possible to perform stochastic crashing by using a stored sample if we can estimate the effect of capacity increments on processing times. If both capacity and workload are modeled by independent lognormal random variables (or even linearly associated ones), then the processing time, given by their ratio, is also lognormal. Under this assumption, it is relatively easy to carry out stochastic crashing calculations. Together with the ability to sequence activities, we can now integrate the traditional CPM tools of sequencing and crashing into PERT 21.

Cates and Mollaghasemi (2007) provide indirect validation for some key ingredients of PERT 21. They describe a safe scheduling application at NASA for the International Space Station program. They use historical data extensively and do not make the independence assumption. NASA is a quintessential project organization, yet historical data proved useful for estimating activity distributions (including correlations) and for feeding simulation models. The simulation results were then used to assess the service level of various project portfolios. NASA decided which sub-projects to take on with the help of simulation based on historical data and made sure that it could deliver on the plan with a reasonable service level. One powerful feature of their simulation is simple: by using their statistical analysis rather than estimated distributions, they avoided the pitfall of GIGO (garbage in, garbage out) that is so often associated with simulation models that rely too heavily on invalidated theoretical models. NASA, however, is not merely a project organization: it is also populated and run by scientists and engineers. Thus, compared to many organizations, NASA is more likely to use advanced management science for analyzing and planning its projects.

The regression analysis described by Cates and Mollaghasemi was performed on an ad hoc basis. That is, they did not write software to perform the historical statistical analysis as part of an enhanced PERT framework (such as PERT 21). Thus, their paper can serve as validation of the basic safe scheduling ideas in projects, but it falls short of presenting a new framework that less sophisticated project organizations can adopt easily. The PERT 21 challenge is to make similar analysis available to less sophisticated project organizations.

In the second part we covered an alternative holding cost calculation framework, Pay As Realized (PAR), which may be appropriate for small projects where some activities do not require dedicated procurement. When supplies are off-the-shelf and labor is from a generic pool, holding costs can be postponed when an activity is delayed beyond its release date. By contrast, our main model can be described as Pay As Planned (PAP), because activities incur costs for the preparations they require starting at their release date, regardless of delay, and purchasing is assumed to involve dedicated materials rather than off-the-shelf ones. Both PAR and PAP models have been developed in the mid-1980s, but until recently they have been studied separately. New results published at about the time our text went to print included useful comparisons between the two approaches and provided effective new PAR solutions for a subset of series-parallel networks, namely converging networks under independent activity durations, for which it was possible to develop optimal algorithms. In presenting these results, we chose to do so in terms of sample-based optimization. We also developed a new solution for a special serial case, with ordered durations. This special case has analytic solutions for both PAR and PAP. We then showed how to apply sample-based PAR solutions for general duration distributions and general project networks. Finally, we showed how to find solutions for mixed models, where some activities are PAP and others are PAR (including the serial ordered case, where we give an optimal solution). One can argue that neither PAR nor PAP is fully applicable to any real project, so mixed models are necessary in practice.

In general, sample-based analysis is always subject to sampling errors but, colloquially, we refer to such models as optimal if they converge to the optimal solution as the sample size grows large; that is, if sampling errors are the only sources of suboptimality. With that in mind, at this stage, except for the ordered case, the extensions we offer are heuristics: we showed how to identify local optima without proof of global optimality. Our focus on sample-based analysis is in line with our objective to provide a feasible practical approach to scheduling problems and, similarly, using heuristics in the absence of proved optimal solutions is a practical necessity. As mentioned above, pure PAR models have been shown to be optimal for converging networks but only subject to independence. We showed that when processing times are ordered, the serial case has an optimal solution that can be identified without iterations. (Similarly, the serial independent case does not require iterations either, but the converging independent case cannot be solved without iterations by any known method.) Now consider the serial model with linearly associated processing times. That case may be viewed as in-between independent and ordered durations. Since we know that in both extremes these models have optimal solutions that we can find by efficient sample-based search, it is conceivable that the same applies for linearly associated durations. An open research question is whether that is true. Similar research questions are whether particular models we offered as heuristics are actually optimal. For demonstrably suboptimal heuristics, if any, it may become interesting to investigate the likely optimality gap and whether it is worthwhile to employ improved heuristics. For instance, our current approach is to start with loose release dates and tighten them towards a local minimum (by matching the gradient to zero), but we can add a search in the opposite direction.

References

Arrow, K.J., T. Harris and J. Marschak (1951), "Optimal Inventory Policy," *Econometrica* 19(3), 250-272.

Arthanari, T. and D. Trietsch (2004) "A Graphical Method for the Pursuit of Optimal or Near Optimal Stochastic Balance," *Proceedings of the 9th International Conference on Industrial Engineering—Theory, Applications and Practice*, The University of Auckland, November, 260-266. Accessible through: <http://ac.aua.am/trietsch/web/Goldratt.htm>.

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* 2(1), 185-203.

Balakrishnan, J. and C.H. Cheng (2000) "Theory of Constraints and Linear Programming: A Reexamination," *International Journal of Production Research* 38, 1459-1463.

Balakrishnan, J., C.H. Cheng and D. Trietsch (2008) "The Theory of Constraints in Academia: Its Evolution, Influence, Controversies, and Lessons," *Operations Management Education Review* 2, 97-114.

Beer, S. (1981) *Brain of the Firm*, 2nd edition, Wiley.

Bock, R.H. (1962) "A New Method for Teaching Linear Programming," *The Journal of the Academy of Management* 5(1), 82-86.

Britney, R.R. (1976) "Bayesian Point Estimation and the PERT Scheduling of Stochastic Activities," *Management Science* 22, 938-948.

Cates, G.R. and M. Mollaghasemi (2007) "The Project Assessment by Simulation Technique," *Engineering Management Journal* 19(4), 3-10.

Chu, C., Proth, J.-M. and Xie, X. (1993) "Supply Management in Assembly Systems, *Naval Research Logistics* 40, 933-949.

Clark, C.E. (1961) "The Greatest of a Finite Set of Random Variables," *Operations Research* 9, 145-162.

Goldratt, E.M. (1990) What is this Thing Called Theory of Constraints, North River Press.

Goldratt, E.M. (1997) *Critical Chain*, North River Press.

Gutierrez, G.J. & P. Kouvelis (1991) "Parkinson's Law and its Implications for Project Management," *Management Science* 37, 990-1001.

Herroelen, W. and R. Leus (2001) "On the Merits and Pitfalls of Critical Chain Scheduling," *Journal of Operations Management* 19, 559-577.

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.

Hopp, W.J. and M.L. Spearman (1993) "Setting Safety Leadtimes for Purchased Components in Assembly Systems," *IIE Transactions* 25, 2−11.

Jansen, S. (2019) "Quantitative Models for Stochastic Project Planning," PhD Thesis, Eindhoven University of Technology, ISBN 978-90-386-4813-2.[*]

[*]*Inter alia*, this thesis covers the material from Jansen et al. (2018) and Jansen et al. (2019), listed below.

Jansen, S., Z. Atan, I. Adan and T. de Kok (2018) "Newsvendor Equations for Production Networks," *Operations Research Letters* 46, 599-604.

Jansen, S., Z. Atan, I. Adan and T. de Kok (2019) "Setting Optimal Planned Leadtimes in Configure-to-Order Assembly Systems," *European Journal of Operational Research* 273, 585-595.

Kelley, J.E. (1961) "Critical-Path Planning and Scheduling: Mathematical Basis," *Operations Research* 9(3), 296-320.

Kumar, A. (1989) "Component Inventory Costs in an Assembly Problem with Uncertain Supplier Lead-Times," *IIE Transactions* 21(2), 112-121.

Leach, L. (2000) Critical Chain Project Management, Artech House.

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.

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.

Millhiser, W.P. and J. Szmerekovsky (2008) "Teaching Critical Chain Project Management: An Academic Debate, Open Research Questions, Numerical Examples and Counterarguments." URL: <http://blsciblogs.baruch.cuny.edu/millhiser/files/2008/10/teaching-ccpm-millhiser-and-szmerekovsky-2008.pdf> (accessed 26 December 2009).

Newbold, R.C. (1998) Project Management in the Fast Lane: Applying the Theory of Constraints, St Lucie Press.

Pittman, P.H. (1994) "Project Management: A More Effective Methodology for the Planning & Control of Projects," Doctoral dissertation, University of Georgia.

Portougal, V.M. (1972). "Constrained Resource Intensity Allocation to Tasks in Project Planning (Russian) *Ekonomika i Mathematicheskie Metody* 9(5), 761-763.

Song, J.-S., C.A. Yano and P. Lerssisuriya (2000), "Contract Assembly: Dealing with Combined Supply Lead Time and Demand Quantity Uncertainty," *Manufacturing & Service Operations Management* 2, 256-270.

Trietsch, D., L. Mazmanyan, L. Gevorgyan and K.R. Baker (2010), "A New Stochastic Engine for PERT," Working Paper.

Trietsch, D. and F. Quiroga (2004) "Coordinating *n* Parallel Stochastic Inputs by an Exact Generalization of the Newsvendor Model," University of Auckland Business School, ISOM Working Paper No. 282 (revised July 2005).

Trietsch, D. and F. Quiroga (2009) "Balancing Stochastic Resource Criticalities Hierarchically for Optimal Economic Performance and Growth," *Quality Technology and Quantitative Management* 6, 87-106.

Trietsch, D. (1993) “Scheduling Flights at Hub Airports,” *Transportation Research, Part B (Methodology)* 27B, 133-150.

Trietsch, D. (2003) "Does Goldratt Understand the 'Theory' of Constraints? Evaporating the 'Do Not Balance' Cloud," Talk presented at INFORMS Atlanta.

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) "From Management by Constraints (MBC) to Management by Criticalities (MBC II)," *Human Systems Management* 24, 105-115.

Trietsch, D. (2005b) "The Effect of Systemic Errors on Optimal Project Buffers," *International Journal of Project Management* 23, 267-274.

Trietsch, D. (2006) "Optimal Feeding Buffers for Projects or Batch Supply Chains by an Exact Generalization of the Newsvendor Model," *International Journal of Production Research*, 44, 627-637.

Trietsch, D. (2007) "System-Wide Management by Criticalities (MBC II): Hierarchical Economic Balancing of Stochastic Resources," *Human Systems Management* 26, 11-21.

Trietsch, D. (2012) "Optimal Crashing and Buffering of Stochastic Serial Projects," Chapter 2 in *Project Management Techniques and Innovations in Information Technology* (John Wang, ed.), IBI Global, 21-32.

Van Slyke, R.M. (1963) "Monte Carlo Methods and the PERT Problem," *Operations Research* 11(5), 839-860.

Wets, R.J.B. (1966) "Programming under Uncertainty: The Equivalent Convex Program," *SIAM Journal of Applied Mathematics* 14, 89-105.

Wiest, J.D. (1964) "Some Properties of Schedules for Large Projects with Limited Resources," *Operations Research* 12, 395-418.

Wollmer, R.D. (1985) "Critical path planning under uncertainty," *Mathematical Programming Study* 25, 164-171.

Yano, C.A. (1987) "Setting Planned Leadtimes in Serial Production Systems with Tardiness Costs," *Management Science*, 33, 95-106.

Yano, C.A. (1987a) "Stochastic Leadtimes in Two-Level Assembly Systems," *IIE Transactions*, 19, 371-378.