Research Notes for Chapter 18*

Following sources and comments for the results in Chapter 18, we provide the proof of Theorem 18.1 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. In our conclusion, 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 18 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 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)

* 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.)

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 18.2 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 (18.9) 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 (to appear) 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 18.1: A Proof

In the chapter we posed Theorem 18.1 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, as blocks. 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 RN18.1: 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.

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

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

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 $\Sigma_{j=1,\ldots,n+1} q_j^* = 1$. This completes the demonstration that the conditions are equivalent to KKT.

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

$$\text{Minimize } Z = E\left[\sum_{j=1}^{n+1}\alpha_j\left(C-r_j\right)\right] = (\alpha+\beta)E(C) - \sum_{j=1}^{n+1}\alpha_j r_j \tag{18.2}$$

Subject to

$$C_j \geq r_j + p_j;\ j = 1,\ldots,n \tag{18.3}$$

$$C_j \geq C_k + p_j;\ \forall k \in P(j),\ j = 1,\ldots,n \tag{18.4}$$

$$C \geq d = C_{n+1} \tag{18.5}$$

$$C \geq C_k;\ \forall k \in P(N) \tag{18.6}$$

Except for the constraint $C \geq d$ (18.5)—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 (18.2) 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 18.2 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 18.2, four out of ten scenarios involve multiple critical release dates (see Table 18.4 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 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 18.2 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 18.1 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 RN18.2: 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 RN18.1, 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 RN18.1, 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 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

[*] 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.

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 central limit theorem (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.

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.

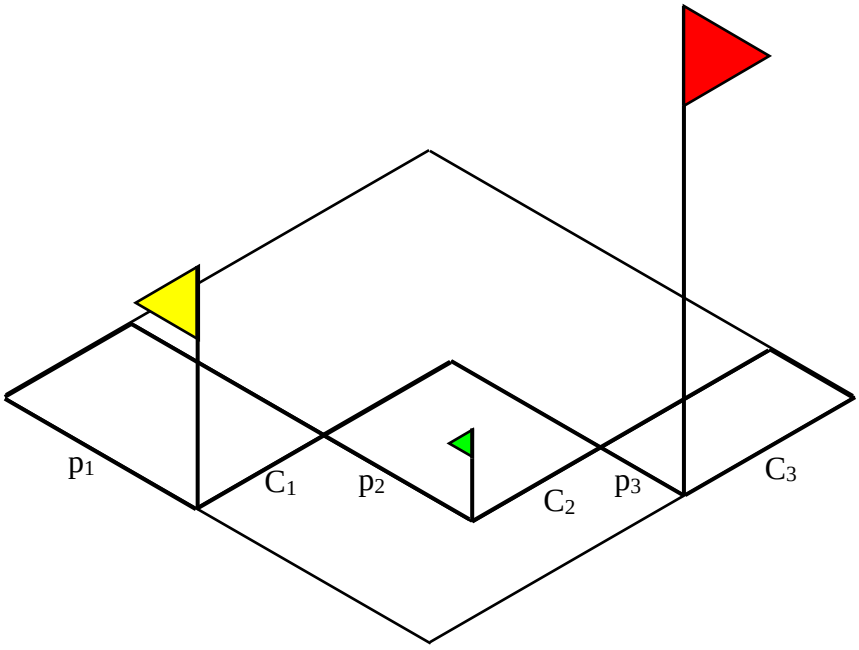

Figure RN18.1: *A Flag Diagram*

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 RN18.1 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

CCPM is generally attributed to Goldratt (1997), but according to Ash & Pittman (1994) 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

* 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.

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 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
2. Manage the binding constraint(s) to improve the objective function maximally[*]
3. Subjugate everything else to the needs of the binding constraint(s)
4. Elevate the binding constraint(s)
5. 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)

[*] 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.

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
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. 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.

---

[*] 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).

The departure point in CCPM is an assumption that activities are typically estimated with a 100% safety time buffer added (that is, the buffer is 50% of the final estimate). Again, this assumption has never been validated, and there is hard evidence that the assumption 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 by Trietsch et al. (2010), the bias was strongly in the opposite direction to Goldratt'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 (the one Goldratt calls "critical chain" and previously named "critical sequence" by Wiest, 1964), 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

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

full feeding buffer. In such cases, Goldratt (1997) 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 reduce the probability 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 (Herroelen, Leus & Demeulemeester 2002). Although CCPM recognizes the need to take sequencing conflicts into account explicitly, the use of 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 used 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

* 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.

† 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 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 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 CCPM 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

[*] 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.

[†] 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.

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 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.

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

* 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 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).

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 considers consuming only 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 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 18.5 and 18.6, 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 18.2.4, where we discuss hierarchy very briefly.) 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.

Conclusion

In Chapters 16, 17 and 18—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). Based on evidence collected after the book was already in print, Trietsch et al. (2010) propose a more general version of the Parkinson distribution. In that version, in a fraction of early activities, the earliness is hidden (so their formal realizations are "on-time"). Preliminary evidence suggests that this distribution is likely to occur in projects. We refer to the text's special case, where the fraction of early activities whose realizations are "on-time" is 100%, as the *pure Parkinson distribution*. In our Research Notes for Appendix A we elaborate on the new generalization and we also discuss the issue of estimation in more detail. 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 45 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.

We did not present PERT 21 as part of our text because it is not mature enough and has yet to prove itself in practice. However, 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.

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.

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). URL: <http://ac.aua.am/trietsch/web/Trietsch&Quiroga.pdf>.

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. URL: <http://web2.cc.nctu.edu.tw/~qtqm/qtqmpapers/2009V6N2/2009V6N2_F2.pdf>.

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. Accessible through: <http://ac.aua.am/trietsch/web/Goldratt.htm>.

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. (to appear) "Optimal Crashing and Buffering of Stochastic Serial Projects," *International Journal of Information Technology Project Management*.

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.