Research Notes for Chapter 17*

Sources and Comments

As we mentioned in our research notes for Chapter 16, most of the useful work in PERT/CPM over the last 35 years—since the publication of *Introduction to Sequencing and Scheduling* (*ISS*)—has been in the area of deterministic sequencing (and thus really enhanced CPM rather than PERT). Whereas the early sources of Chapter 17 date back to *ISS*, some of them repeated in our Research Notes for Chapter 16, subsequent results, appear in Demeulemeester and Herroelen (2002). Readers who are interested in optimization methods for the resource constrained project scheduling problem are specifically referred there: they provide details for numerous branch and bound and integer programming models for this purpose. Among other things, they also describe and critique the Critical Chain methodology—which they call Critical Chain Buffer Management (CCBM)—and they include some stochastic models. Beyond those, the chapter itself cites the main new sources that we used. However, for a recent broad survey of related models, see Hartmann and Briskorn (to appear), and for a contemporary description of existing stochastic counterpart sequencing models see Ballestín (2007). (We mention some of his own results later, where they fit in our context, but we do not duplicate his summary of previous results. Nor do we repeat the vast majority of important citations that appear in the other two sources.)

Advanced heuristics are based on generic neighborhood search methods such as tabu search, simulated annealing and genetic algorithms (GA). Kolisch and Hartmann (2006) provide an extensive comparison among such algorithms. In general, the ranking of these algorithms changes over time—to wit, Hartmann and Kolisch (2000) give a slightly different ranking—but, as a rule, advanced neighborhood search techniques—including GA—provide excellent results. Surprisingly, the modified search approach (which can be combined with such generic methods) has not been studied until recently. The earliest source of that idea that we know of is Fleszar and Hindi (2004), who also recommend variable neighborhood search. They use a modified all insertion (AI) neighborhood and report excellent results relative to former benchmarks. Recall that in the chapter we showed how to modify the simpler adjacent pairwise interchange (API) neighborhood. However, as we showed in detail in our Research Notes for Chapter 4 (see *Adapting Search Heuristics to the Solution of* $1 \mid prec \mid \Sigma w_j T_j$), modified insertion can be achieved by a series of modified API steps. Likewise, any modified nonadjacent pairwise interchange can be achieved by two modified AI steps.

In our Research Notes for Chapter 4 we also observed that genetic algorithms tend to generate feasible sequences; that is, offspring inherit feasibility from their parents, unless a mutation violates it, and therefore it makes sense to allow only mutations that

---

belong to a suitably modified neighborhood. Here, we stress one point again: GA had been reported as a highly competitive algorithm for projects using research results that, as a rule, did *not* rely on modified neighborhood search. Thus, an important advantage of GA may be diluted when it is compared to other techniques using modified neighborhood search. In effect, it is now necessary to re-test all former approaches, but with modified neighborhoods. Among other things, this re-testing will show whether GA (with modified mutations) retains its previously reported advantage.

Renewable, Nonrenewable and Doubly-Constrained Resources

Resources of the type we considered in the chapter are available throughout the project duration and thus they are not depleted during the project. When released from their current activities, such resources are—in a sense—renewed, so they are also known as *renewable* resources. By contrast, some resources—such as the project budget—are *nonrenewable* because once consumed they are no longer available. A *doubly constrained* resource is limited each period and also in terms of total consumption. In practice, budget constraints are usually doubly constrained, which means that funds flow rate and the total funds available are both limited.

Typical models address nonrenewable constraints (e.g., budgets) as if exceeding them is simply impossible. That makes sense under the deterministic assumption, which most of those models make. In reality, however, processing times are always stochastic and the penalty for exceeding such constraints may be different; e.g., in the budget case, additional funds are often found, but at a cost (sometimes having the effect of throwing good money after bad). Of course, project managers' careers can be hurt by budget violations, and therefore they actively try to avoid them. Considering that time is also a nonrenewable resource, a similar comment applies to due dates and *deadlines*. Although some sources use the terms "deadline" and "due date" interchangeably, others interpret the former as a stricter limit: deadline violations are equivalent to project failure, whereas due date violations may be penalized, but it is accepted that they are not always avoidable. A recent example of a strict deadline is the stadium construction project for the Olympic Games in Athens. That particular project barely met the Games' opening ceremony schedule, and incurred serious costs for the emergency mode of operations that was required as a result. On the one hand, we use due date models for scheduling—which is part of planning—whereas control actions are taken during execution. On the other hand, the likely control cost of meeting a deadline that is liable be violated without intervention typically increases with the expected looming tardiness. If so, the appropriate scheduling (planning) model is still the use of due dates with an increasing tardiness penalty. Specifically, our safe scheduling models can include the expected control cost as a key part of the penalty cost function.

On Sequencing vs. Resource Leveling

Following the pioneers of project scheduling theory, such as Kelley (1963) and Wiest (1964, 1967), we address the resource-constrained project scheduling problem by an explicit sequencing approach. The same approach has been taken by subsequent sources, such as *ISS*, Morton and Pentico (1993), and Demeulemeester and Herroelen (2002). What's common to all of them is that although they address a practical problem (for instance, Wiest 1967 describes a commercial software implementation), they

approach the subject from a theoretical/normative point of view. In this approach, it is implicit that we must add disjunctive precedence constraints to the ones given by the original project network. In the chapter, we referred to them as *soft* precedence constraints. As a result, the critical path must now reflect not only the original (conjunctive) precedence constraints but also the disjunctive ones. Because the critical path depends on both conjunctive and disjunctive constraints, Wiest (1964) proposed changing its name to the *critical sequence*. This proposal has not been adopted, at least not widely, and indeed there is no need to rename the critical path.[*] Typical project management texts, however, address the issue by describing outdated practice instead. In this particular instance, early practice—possibly because early project scheduling software, including the original PERT software, did not include sequencing modules at all—adopted the inferior MRP sequencing approach of leveling resource usage instead of sequencing resources in a straightforward manner. (We discussed MRP briefly in our Research Notes for Chapter 12; see *The Role of JIT as Compared to MRP*.) Thus, typical project management texts, and other authoritative sources such as *Project Management Body of Knowledge* (PMBOK)—published by the Project Management Institute—address *resource leveling* instead of sequencing. The objective of resource leveling, as the term suggests, is to keep the resource utilization profile as flat as possible without violating the maximum resource constraint. Because resource leveling is pursued by making sequencing decisions, one might think that the difference is slight. Nonetheless, the resource leveling approach—when used manually or interactively—is less likely to lead to good results.[†]

A schedule with level resource usage is indicative of a good sequence (because there is little or no idling) but it is best pursued indirectly by effective sequencing. True, if we generalize the problem that we discussed by treating resource capacities as decision variables, then the resource leveling problem might lead to useful results. The idea is to compare the total economic implications of running the project with various resource capacity profiles and to choose the best one. However, even when the true objective is to decide how many resource units to allocate to the project, it is better to follow the approach of Example 17.2, where we studied the relationship between capacity and makespan. That is, we can select the optimal capacity by finding the best combination of capacity and makespan. But for any tentative capacity level, we can and should use explicit sequencing models to minimize the makespan. For a given resource capacity that is dedicated to the project from start to finish, minimizing the makespan automatically minimizes the total unused capacity, which is the desired implicit objective of resource leveling.[‡]

---

[*] The term *critical chain* has been proposed for the same purpose much more recently, with a wider reception. Except for commercial marketing purposes, however, that renaming is still a redundant exercise.

[†] On the bright side, typical project scheduling software, at least since the package described by Wiest (1967), follows the correct explicit sequencing approach. It is still far from ideal, however. In that context, our comment in the chapter that the performance of the better software packages is comparable to using the LFT priority list is due to Herroelen (2005). To our knowledge, that observation is still valid.

[‡] When resources can be inducted during the progress of the project and released towards the end, as in the skyline profile, the objective of maximizing utilization becomes more complex. To our knowledge, that model has not been studied yet, but it would be straightforward to solve it by neighborhood search methods. Incidentally, this objective is similar to minimizing total holding costs, our next subject. A related new research area is to balance the project's dedicated resource level with outsourcing or temporary hire of

Minimizing Total Holding Costs

In Chapter 18 we introduce a stochastic scheduling model designed to minimize the sum total of expected holding costs and tardiness penalties. Such a model is necessary for setting optimal release dates within a safe scheduling context. For that purpose we assume that sequencing decisions have already been made and are reflected by soft precedence constraints; that is, we only address the pure scheduling model. Here, we address the deterministic counterpart sequencing model.

Suppose a project consists of n activities and has a due date (d). Each activity incurs an earliness cost per unit time denoted α_j for activity j. This earliness cost reflects the economic value of postponing the activity and may also be viewed as a holding cost. The project incurs a tardiness cost per unit time denoted β. In practice, tardiness cost reflects the cost of delay in obtaining revenue and often includes explicit compensation to customers when a due date is missed. The fixed cost element that we introduced in Chapter 16, c_f, is part of this tardiness cost per unit, because it has to be borne until the project completes. To this element we add any explicit or implicit tardiness penalties. Ideally, we would like to balance activities' earliness costs against the project's tardiness costs. The objective is thus to minimize total E/T cost, or

$$Z = \beta(C - d) + \sum_j \alpha_j(C - r_j)$$

where $C \geq d$ represents the project completion time. We assume that d is given, as if it had been negotiated with a customer. The customer provides no incentive for early completion, so we proceed as if the project is delivered to the customer at the due date or as soon as possible thereafter. In other words, if the project completes prior to the due date, delivery to the customer is on the due date, fulfilling the negotiated agreement. If the project completes later than the due date, then the tardiness penalty applies. Earliness cost can be viewed as weighted flowtime, or the holding costs incurred while an activity is held in the system. In the project setting, the flowtime for activity j is given by $(C - r_j)$, because once started, the activity becomes part of the project and is released from the system only when the entire project is complete. In the objective function, this flowtime is multiplied (weighted) by α_j. For a given sequence, and deterministic activity times, that objective is minimized by starting each activity at its latest start (as calculated *after* including the disjunctive constraints in the network). As summarized by Demeulemeester and Herroelen (2002), Vanhoucke, Demeulemeester and Herroelen (2001) address a more general case where each activity has its own due date and tardiness rate. (Individual activity due dates are appropriate when they reflect important milestones; e.g., the customer may make progress payments upon such milestones.) They show that setting release dates in this case can be solved in polynomial time. In the stochastic case, we

---

additional resources on an ad-hoc basis. As a rule, outsourcing or temporary hiring costs more per time unit than it would to dedicate the additional capacity to the project. However, if we dedicate that capacity, we must pay for it throughout the project duration. The balance is obtained when hired resources are only used for sufficiently short periods to justify their higher rates. This is akin to our models for minimizing the due date plus weighted expected tardiness. The cost of dedicating resources until the due date is analogous to the size of the dedicated capacity (for which, effectively, we pay in advance), whereas the expected cost due to tardiness is analogous to hiring or outsourcing costs (for which we pay only when necessary, but at a relatively high rate).

show how to address the basic problem (without individual activity due dates) in Chapter 18. The stochastic counterpart of the more general problem—with individual activity due dates—can be addressed by an essentially equivalent approach. Those models are also polynomial and tractable, although they require a stored sample as part of the input (that is, they are polynomial in input size but that input size tends to be large). However, the sequencing model is NP-hard (because it generalizes known NP-hard problems).

Here, we propose a particular list as a seed for list scheduling heuristics. The list is logically feasible so it can be used with both the parallel and serial approaches. It can also be used as a seed for a biased random search. To obtain this list, we solve a single machine version of the problem where all activities require the same resource. Because the makespan is a constant, there is no need to consider tardiness; that is, the project is scheduled to complete at the due date or, if that's impossible, the project starts at time zero and the constant tardiness penalty is ignored (or, equivalently, the due date is reset to the makespan). We can solve the problem optimally in polynomial time if the precedence constraints are series-parallel. One way to do so is to follow the template of Section 8.3.3, but reverse the timeline first and use SWPT instead of SPT to form strings and to sequence parallel activities. That is, reverse all precedence constraints, and sequence in the reverse direction by an analogous approach to that of Section 8.3.3, but with weights. Equivalently, we can solve in the forward direction if we use LWPT instead of SPT. To elaborate, we show how to modify Algorithm 8.3. That is, we temporarily assume a parallel chain precedence structure. For our purpose, string u has a value p_u/w_u associated with it, where p_u is the total processing time of the activities in the string and w_u is the sum of all earliness costs, α_j, of those activities.

ALGORITHM RN17.1
Parallel Chain Algorithm for Project Weighted Flowtime

Step 1. Initially, each activity is a string.

Step 2. Find a pair of strings, u and v, such that u directly precedes v and $p_v/w_v \geq p_u/w_u$. Replace the pair by the string (u, v). Then repeat this step. When no such pair can be found, proceed to Step 3.

Step 3. Sort the strings in non-increasing order of p/w.

The generalization to the series-parallel structure is then analogous to the one described in the text for the unweighted case. If the network is not series-parallel, the same approach can still be used as a heuristic. However, in that case, as we create strings, we may prevent other potential strings from being considered later. For instance, consider the interdictive graph (Figure 16.6 or RN16.6). We might wish to combine activities A and C into a string (if $p_C/w_C \geq p_A/w_A$) but we might also wish to combine A and D (if $p_D/w_D \geq p_A/w_A$), and we can't choose both at the same time. To see the complexity that ensues, observe that if we sequence C before D, we obtain a series-parallel network with A and C in series, together in parallel to B, and that subnetwork is in series with a subnetwork

comprising *D* and *E* in parallel to each other. If we sequence *D* before *C*, we obtain another series-parallel network with *A*, *D* and *C* in series, together in parallel to *B* as one subnetwork in series with *E*, the complementary subnetwork. We cannot tell in advance which of the two options leads to the optimal solution. Therefore, unlike the series-parallel case, the order in which we proceed is important and the solution is no longer guaranteed to be optimal. If we respond by considering all possible options, the complexity of generating the list becomes exponential (and depends on the number of imbedded interdictive graphs in the network). An obvious greedy heuristic rule is to select by LWPT; for instance, if $p_C/w_C \geq p_D/w_D$, we sequence *C* before *D*. This step should be preceded by first forming all strings that can be formed by Step 2 of Algorithm RN17.1 within all chains imbedded in the network. Parallel subnetworks should also be resolved where possible.

Now consider the stochastic case, where we draw on the similarity between the parallel machine model and project scheduling. Recall that in the conclusion of Chapter 9 we observed that the objectives of makespan and flowtime are in conflict. For instance, when processing times are exponential—which we assume henceforth to simplify the exposition—the former is minimized by LEPT and the latter by SEPT. But when we count flowtime only from the actual start of each job until completion of the last job, as we would do in our current model, then the two objectives are no longer in conflict. Now we achieve both of them by LEPT, and the use of LWEPT should be reasonably robust for the weighted case. Hence our list—which essentially is an adaptation of LWEPT for the case of precedence constraints—is likely to be robust for the more realistic stochastic case. One caveat that we must state is that part of the effectiveness of LEPT in the stochastic parallel machine case is predicated on the use of dispatching. But dispatching is not always possible in the project case, depending on practical considerations that are not always reflected by the project network. For instance, there may be a need to plan exactly which resources will be allocated to each activity and when. Then, it is often necessary to stage these resources in advance, while predecessor activities are still in progress. Thus, postponing an activity that was due to start soon can waste preparations already made for it, whereas starting another activity immediately may simply not be possible. One practical response is to define a *frozen horizon* period, during which changes to the current sequence are not allowed.

Net Present Value

An alternative to measuring the economic value of holding costs is the *net present value* (NPV) approach. Minimizing NPV is especially appropriate for projects with very long durations. In such cases, the time value of money has to be taken into account: funds spent early are more expensive (Demeulemeester and Herroelen, 2002). This approach may also be viewed as a generalization of our holding cost model.

On the Performance of Tie-Breakers and List Sequencing Procedures

Whereas the performance of our proposed list for minimizing holding costs has not been studied yet, some related results are known. Tigranyan (2008) compares the performance of two list heuristics—LST (without dynamic updating) and LFT—with or without tie-breakers for deterministic problems with 30, 60 and 90 activities. The problems are taken from the PSPLIB resource-constrained project problems library

(Kolisch and Sprecher, 1996). Tigranyan pursues the makespan objective, but he also compares the flowtime of various lists and tie-breakers. He reports that the use of LPT as a tie-breaker in the forward direction tends to reduce total flowtime. Likewise, when list scheduling is applied in the reverse direction, SPT is the tie-breaker that tends to reduce total flowtime. Interestingly, he also shows that, for those problems in PSPLIB, there is a significant advantage in reverse-sequencing. The best combination for total flowtime minimization is LFT with SPT as the tie-breaker but in the reverse direction. The best makespan, however, is obtained by LFT with LPT as the tie-breaker, again in the reverse direction. In the forward direction this combination tends to reduce total flowtime relative to other tie-breakers. But in the reverse direction, the use of LFT with SPT as the tie-breaker—that is, the best combination for the purpose of reducing total flowtime—was not the best in terms of makespan. This result should not be surprising if we recall from Chapter 9 that using SPT for parallel machines is optimal for flowtime but counterproductive for makespan reduction. One inference might be that the PSPLIB problems are not symmetric with respect to reversal of the timeline. Indeed, there is no need to assume that practical project networks are symmetric either, but there may be performance differences between lists that don't occur in general but rather depend on the library. A practical conclusion is that it is worthwhile to try various combinations in both directions. As yet another viable alternative, Demeulemeester and Herroelen (2002) also recommend sequencing from both ends towards the middle.

Tigranyan (2008) also confirms an observation made earlier by Kolisch (1996)—for the same PSPLIB problems—that the parallel approach tends to outperform serial sequencing when the list is used just once. However, Hartmann and Kolisch (2000) report that when multiple sequences are generated by biased random search, the serial approach outperforms the parallel approach. That reversal may be explained by the fact that in some instances the optimal solution is active but not nondelay, so only the serial approach can ever yield the optimal solution. In the cases tested by Tigranyan (2008), about 15% of the best active sequences identified required delay. Apparently, when multiple sequences are tested, the ability to generate active sequences with delay more than compensates for the inferiority of the serial approach in the single run case. Tigranyan also found that when the list is run just once for those PSPLIB problems, using the parallel approach, LST (without dynamic calculation) outperforms LFT in the forward direction although LFT outperforms LST in the reverse direction. But when the serial approach is used, LST outperforms LFT in both directions. Whereas the differences are slight (and were not tested for statistical significance), this is still a surprising result. However, according to Hartmann and Kolisch (2000), when biased random search is performed, using the serial approach but with multiple runs, LFT is the distinct best seed. Again, the practical conclusion is that it is worthwhile to test more than one list in more than one direction.

When processing times are stochastic, the use of the parallel approach is straightforward—effectively, it guides real-time dispatching as the project progresses. As in the deterministic case, the schedule is then nondelay and thus active. However, the serial approach can no longer guarantee active schedules. To see that, consider that an activity that can fit earlier than its priority calls for in the deterministic case may not be allowed to start early in the stochastic case if there is any likelihood it will postpone a higher priority activity. That is, no low-priority activity should be fit ahead of a high-

priority one if its maximum exceeds the gap available for it, but it might have fit in the deterministic case. (As a rule, we recommend using lognormal activity distributions. If so, the maximum is not bounded and we can *never* fit an activity earlier than its priority demands.)

An alternative implementation of the serial approach suited to real-time use (that is, dispatching mode) has been used for both deterministic and stochastic analysis. Recently, it was studied by Ballestín (2007), who refers to it as the *stochastic serial schedule generation scheme*. Under this stochastic serial approach, we use dispatching but we only consider activities in the list order. That is, if activity *i* precedes activity *j* in the priority list, *j* cannot be dispatched before *i* (although they can be dispatched at the same time when resources permit). If *j* becomes available before *i*, it must wait. Thus, this approach allows idling, as in the deterministic serial approach, but it does not allow using any gaps that may have been created earlier in the schedule (because, in practice, we would not be able to tell in advance whether they are sufficiently large). Based on experiments using stochastic versions of PSPLIB problems with 120 activities, Ballestín reports a small advantage for the stochastic serial approach over the parallel one (but provides no information as to whether that difference is statistically significant). Recall that in the deterministic case we noted an advantage to the serial approach *if* many neighboring lists are used (otherwise, the advantage goes to the parallel approach). Indeed, Ballestín's tests use either biased sampling or GA, both of which involve multiple lists, and thus his observation for the stochastic case reinforces the one we made in the deterministic case. A more important recommendation provided by Ballestín, however, may be interpreted as a step towards the parallel approach in the sense that it promotes starting activities earlier than would be the case under the pure stochastic serial approach. He recommends first using the priority list to generate a deterministic counterpart schedule (with the serial or parallel approach) and then redefining the list to reflect the actual order in which activities are performed for that list. He reports that such redefined lists generate much better results. He also reports that using a good deterministic counterpart sequence is a robust heuristic. More precisely, he reports that it outperforms stochastic analysis when variation is low or moderate, and the results indicate that it performs quite well even when variation is high. That stochastic analysis is based on using simulation to judge the quality of each (redefined) list generated either by biased sampling or by GA. Nonetheless, it is important to take into account the Jensen gap, which, in his experiments, turned out to be much higher than previous studies suggested. This observation supports our own recommendations to use the deterministic counterpart sequence but account for variance and the Jensen gap explicitly (see Section 11.5—specifically Figure 11.5—and Chapter 18).

Combining Crashing and Sequencing Decisions

Because academic papers tend to focus on basic models, in general, not much work has been done on combination problems such as considering the effects of crashing and sequencing together. The little work that was done in this field assumes that crashing is discrete, rather than continuous as in the original CPM formulation. That is, each activity has one or more possible *modes*, each of which has a different cost and a different duration. The resulting multimode resource-constrained project scheduling problem (MRCPSP) can be formulated as an integer program so it can be addressed by generic

solution approaches such as branch and bound, Benders decomposition, and neighborhood search heuristics. By contrast, if we were to specify continuous crashing instead, we would obtain a relaxation that could be solved by mixed integer programming but is less directly amenable to neighborhood search heuristics. Perhaps for this reason this version of combining continuous crashing with sequencing has not been studied yet. We note in passing that, in principle, a Benders decomposition approach can still be pursued for mixed integer programs, and could thus be tested for the continuous crashing version of the RCPSP. Although this approach is often slow, it can sometimes be accelerated by using problem specific features. For instance, Hazir et al. (2009) report solving a bi-modal crashing model *without* including sequencing decisions—where each activity has just two nodes and, except for a budget constraint, only conjunctive constraints apply—for up to 100 activities by using Benders decomposition with such streamlining. That bi-modal problem is one of several essentially equivalent formulations of the most basic discrete crashing model.

In general, in the multimodal case, the set of modes considered has progressively shorter durations and higher costs. (Any mode that does not fit that profile can be safely removed from consideration.) Considering that the state of the art solution of the discrete crashing problem, even without including sequencing decisions, can only address up to 100 activities, clearly the full-fledged problem, with multiple modes and sequencing decisions, has not been solved yet for realistic instances, and only heuristics have been attempted. Even heuristics have not yet been effective for more than 30 activities, each with three modes. Hartmann (2001) addresses the problem by GA and compares the results to other approaches, including the use of branch and bound approach with a time limit. He reports much better results for the GA. We should also repeat the observation from our Research Notes for Chapter 4 that there are many details in the application of heuristics that make comparisons difficult and call for care in interpreting labels such as GA. In this instance, Hartmann demonstrates that his particular GA application, with particular streamlining features, performs better than other approaches, including at least one earlier GA application.

Whereas the ideal approach would solve for the optimal sequence and modes together, in practice we may safely expect that heuristics will remain the norm. We have already reached a similar conclusion even without considering multiple modalities. The most accessible heuristic, both for continuous crashing and the multimode model, is simple: solve the two problems separately. But there is an important observation that applies in this case: crashing (discrete or continuous) should not be considered before sequencing. If we do it the other way around, that is, if we ignore sequencing for a while, make crashing decisions first and only then make sequencing decisions, we are liable to crash activities that are not even critical, let alone the best crashing candidates. We are also liable to increase the resource usage by crashed activities without regard to resource availability; that is, we may choose crashing options that are not even feasible. For those reasons, it is strongly recommended to sequence first. However, it is possible to revisit sequencing decisions after a complete round of sequencing and crashing has taken place. That is, we can engage in a cyclical application of sequencing and crashing (Trietsch, 2005).

Approximate Branch and Bound

Branch and bound options include adapting the shifting bottleneck algorithm. This requires extending the single-machine head-body-tail model to parallel machines. However, the bounds created this way are not as effective as those that apply for the single machine case (see Demeulemeester and Herroelen, 2002). In addition, as in the job shop case, the shifting bottleneck logic can be used in a heuristic search. Morton and Pentico (1993) discuss shifting-bottleneck heuristics extensively. They use a branch and bound structure in the forward direction. To estimate the remaining time of a partial schedule, they use a queueing model approximation (although they assume deterministic times and apply the stochastic queueing formulas on an ad hoc basis). With such estimates in place, they then branch in the most promising direction. By virtue of using queueing models, the most loaded machine downstream from any given partial schedule contributes the highest estimated total queueing and processing times to these estimates. By branching in the most attractive direction, in the forward direction, it becomes possible to obtain a partial sequence that can be used immediately and updated later. Therefore, this approach is especially attractive in a highly stochastic environment where activities require preparation, because it allows creating stable sequences for the near future while maintaining flexibility beyond the frozen horizon. Key is the use of estimates to approximate the effects of current decisions into the future without actually trying to precisely anticipate the future too far in advance. This is a promising approach, especially for stochastic models, but it requires further research. Specifically, queueing models that involve complex resource sharing models are virtually nonexistent, but they would be helpful in this context.

Robust Scheduling

In our Research Notes for Chapter 7 we discussed robust scheduling in generic terms. In the project scheduling context, robust scheduling models typically seek base schedules that anticipate subsequent reactive scheduling and aim to avoid disruptions as much as possible (Demeulemeester and Herroelen, 2002; Herroelen and Leus, 2005). At the time of this writing, there is no evidence that such models have been implemented in practice. To some extent, robust scheduling and safe scheduling aim to resolve the same practical problem. Under safe scheduling we achieve robustness by incorporating appropriate safety time cushions in the schedule. When safety time buffers are used, the need for reactive scheduling is reduced and the buffers also provide time to actually plan reactive actions when needed. As another option, we noted that approximate branch and bound algorithms of the type recommended by Morton and Pentico (1993), because they estimate the downstream performance of a partial schedule by queueing models, may be suitable for the production of good sequences to be used during a frozen horizon period. Under that approach—which lies in between static sequencing and dispatching—sequencing decisions are simply not made too far into the future, thus alleviating the need for explicit preparation for future sequence changes by reactive scheduling. Yet, firm decisions are being made for the near future (the frozen horizon) and tentative decisions are made for the medium future (the period beyond the frozen horizon for which we prepare a static schedule but we do not consider it final until required).

References

Baker, K.R. (1974) *Introduction to Sequencing and Scheduling*, Wiley, Hoboken, NJ.

Ballestín, F. (2007) "When it is Worthwhile to Work with the Stochastic RCPSP?" *Journal of Scheduling* 10, 153–166.

Demeulemeester, E.L. and W.S. Herroelen (2002) *Project Scheduling: A Research Handbook*, Kluwer Academic Publishers.

Hartmann, S. (2001) "Project Scheduling with Multiple Modes: A Genetic Algorithm," *Annals of Operations Research* 102, 111-135.

Hartmann, S. and D. Briskorn (to appear) "A Survey of Variants and Extensions of the Resource-Constrained Project Scheduling Problem," *European Journal of Operational Research*. URL (accessed 9 November 2009): www.hsba.de/de/pdf/professoren/hartmann/models.pdf

Hartmann, S. and R. Kolisch, (2000) "Experimental Investigation of State-of-the-Art Heuristics for the Resource-Constrained Project Scheduling Problem," *European Journal of Operational Research* 127, 399-407.

Hazir, Ö, M. Haouaric and E. Erel (2009) "Discrete Time/Cost Trade-off Problem: A Decomposition-based Solution Algorithm for the Budget Version," *Computers and Operations Research*, doi:10.1016/j.cor.2009.06.009.

Herroelen, W. and R. Leus (2005) "Project Scheduling under Uncertainty: Survey and Research Potentials," *European Journal of Operational Research* 165, 289–306.

Kelley, J.E. (1963) "The Critical Path Method: Resources Planning and Scheduling," Chapter 21 in J. Muth and G.L. Thompson *Industrial Scheduling*, Prentice-Hall, 347-365.

Kolisch, R. (1996) "Serial and Parallel Resource-Constrained Project Scheduling Methods Revisited: Theory and Computation," *European Journal of Operational Research* 90, 320-333.

Kolisch, R. and S. Hartmann (2006) "Experimental Investigation of Heuristics for Resource-constrained Project Scheduling: An Update," *European Journal of Operational Research* 174, 23-37.

Kolisch, R. and A. Sprecher (1996) "PSPLIB – A Project Scheduling Problem Library," *European Journal of Operational Research* 96, 205-216.

Tigranyan, G. (2008) "Resource Constrained Project Scheduling Problems," Master Thesis, Industrial Engineering & Systems Management, College of Engineering, American University of Armenia, Yerevan, Armenia.

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.

Vanhoucke, M., Demeulemeester, E. and Herroelen, W. (2001) "An Exact Procedure for the Resource-Constrained Weighted Earliness-Tardiness Project Scheduling Problem," *Annals of Operations Research* 102, 179-196.

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

Wiest, J.D. (1967) "A Heuristic Model for Scheduling Large Projects with Limited Resources," *Management Science* 13, B359-377.