Research Notes for Chapter 1[*]

One of our guiding principles when organizing the main text was to go from simple to complex topics and maintain technological order—that is, basing each new result only on previously covered results. A secondary guiding principle, was to minimize any discussion of connections between models, thus maintaining a simple structure. The subject area we describe, however, did not develop along the same lines, and therefore it may be helpful for research-oriented readers to look at the bigger picture without adhering to these principles with the same fidelity. One purpose of our research notes is to provide such a discussion. Another purpose is to consider research issues chronologically, extending speculatively into the future. In that spirit, we discuss some seminal historical developments, and we mention open research questions that should be addressed by future research. We also cover topics that are not sufficiently central to justify inclusion in the text itself. Finally, to prepare research-oriented readers to read original source material, we occasionally need to discuss issues that do not appear in the main text but are likely to be encountered in the literature. When this is the sole motivation, we do not provide in-depth coverage or an exhaustive reference list. Typical papers that would require such detail also provide the necessary references. In the research notes of this chapter, we discuss some historical developments and discuss computational complexity in more detail.

The issue of sequencing must have been on the agenda ever since mankind became involved in large projects, at the latest. But there were probably no formal models involved with those endeavors. The first modern scheduling tool, the Gantt chart, is due to Henry L. Gantt, and was developed during World War I. But mathematical models for scheduling took off only after the publication of a seminal paper by Johnson (1954) on the optimal sequence of jobs on two consecutive machines (an arrangement known as a *flow shop*) to minimize the *makespan*, defined as the time from the start of the first job until the completion of the last job. Johnson provided a simple rule for this purpose (which we cover in Chapter 10) but he also noted that with some exceptions, the three machine problem was already much more complex. (Indeed, today we know that the three-machine flow shop makespan minimization problem is NP-hard.) Soon thereafter, another seminal paper, Smith (1956), captured in one place several simple sequencing models—including Johnson's rule—all based on sorting the jobs and sequencing accordingly. (We discuss most of these models in Chapter 2.) Although both authors also addressed problems that resisted their own simple rules, these two papers still encouraged the thought that it should be possible to solve many more models efficiently and elegantly, and much energy was devoted to this end, with a handful of notable successes but also with a lot of frustration: it became increasingly clear that most

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

Citation details: Baker, K.R. and D. Trietsch (2009) Research Notes for Chapter 1 in *Principles of Sequencing and Scheduling* (Wiley, 2009). URL: http://mba.tuck.dartmouth.edu/pss/.

scheduling models were not easy to solve efficiently. This state of affairs characterized the first 25 years of the development of scheduling theory.

With few exceptions, the models addressed during this period ignored the uncertainty that is so prevalent in practical situations. They also assumed that finishing a job early is always at least as good as finishing it on time. Ignoring uncertainty in the early models was not the result of failing to see that it exists. Right from the start, it was clear to the pioneers of scheduling theory that practical scheduling challenges involve combinatorial complexity (of the type addressed by Johnson and Smith) and the complexity associated with uncertainty. In general, one way to address uncertainty—and the way we proceed in the text—is to use stochastic models. Stochastic models assume that the uncertainty can be addressed by providing probability distributions for key parameters in models that are otherwise deterministic. Such distributions are ideally estimated from historical data. Nonetheless, there are situations where historical data is sparse or missing but we can still recommend the stochastic approach. This type of analysis is based on Bayesian statistics, one of whose basic tenets is that the very fact that a decision must be made is tantamount to the use of subjective distributions by the decision makers, and it is better to do so explicitly. Interested readers can find out more about the theory of decision-making under uncertainty and the use of subjective distributions for this purpose (if necessary) in Raiffa and Schlaifer (1961). But because deterministic models (which addressed combinatorial aspects) were already very challenging, practically all the work concerned deterministic models. Only the most tractable stochastic models were addressed, reflecting a tiny percentage of research publications of that era; e.g., see Rothkopf (1966). Furthermore, during the first 25 years all models had *regular* performance measures. A performance measure is regular if it cannot be improved without making at least one job finish earlier; i.e., there is no incentive to delay a job unless this allows another job to finish earlier. For example, the makespan—Johnson's performance measure—is regular, and so were all the models addressed by Smith. Thus, when stochastic scheduling emerged as a major research direction, around 1980, the deterministic arsenal was all about regular models.

This state of affairs was not due to negligence or ignorance. Based on the analytical scientific approach, it was rational to address combinatorial complexity first, by deterministic models and postpone stochastic analysis. Indeed, there is evidence suggesting that the reason stochastic models were rarely addressed during the first 25 years is directly related to the magnitude of the challenge to solve even deterministic and regular models analytically. The evidence lies in the fact that PERT (program evaluation and review technique) was introduced in the late 1950's, and it involved sophisticated stochastic analysis right from the start. To appreciate how sophisticated this analysis had been, see not only Malcolm et al. (1959) but also Clark (1961)—a source that showed how to approximately calculate the moments of a project makespan distribution in spite of the complex dependencies that are involved. (The PERT team chose not to include this model in the original paper for simplicity, but they did mention its existence). They could do that because sequencing decisions were not involved. When sequencing issues were subsequently studied in the context of PERT—see Wiest (1964)—deterministic models were employed there, too. As a result, to this day, there are project related papers that address stochastic issues, some of which also address safety, and there are project related papers that address sequencing, but we know of no deep analysis of the two issues in combination. Meanwhile, PERT aside, stochastic issues were addressed by two major and somewhat overlapping approaches. One, by queueing theory, started (in this context) with the work of Jackson (1957), who modeled a job shop as a network of M/M/1 queues. The other was by simulation.

Queueing theory put very little emphasis on sequencing effects, and the main results showed how variation is transformed to waiting time. The simplest queueing models—those that generalized the M/M/1 model to job shops—could also be used to calculate distributions of waiting times, but more general models could rarely achieve that.

For the study of sequencing issues, simulation (of the type we cover in Chapter 15) was perhaps more successful. At the start, a job set was generated randomly and stored, complete with arrival times, processing times, due dates and routing information. For this stored set, various dispatching rules were used and the results compared in terms of the fraction of jobs tardy and the mean tardiness. For example, it was discovered that choosing the shortest job first frequently yields few tardy jobs but tardy jobs are liable to be very tardy). Such simulations are used to this day to compare heuristic dispatching rules, and their practical value lies in the fact that sequencing is often carried out in practice precisely by such dispatching rules. With few exceptions, however, once the job set had been determined, these simulations treated the job descriptors as deterministic. So some aspects of randomness escaped the analysis. For example, one cannot sequence by the final processing time but only by its distribution (e.g., by its mean) because the precise processing time is not known before processing has taken place. Thus, in a technical sense, these simulation studies left a lot to be desired, but they still provided the most practical results of scheduling theory at that time.

To summarize, by the time the field turned 20, a lot of work had been done on deterministic models. From PERT, we learned some important lessons about stochastic scheduling (without sequencing decisions); from simulation we learned the systemic effect of simple dispatching rules; and over the years we have also acquired an impressive arsenal of highly theoretical deterministic scheduling models. It was not until then that the NP-complete equivalence class was discovered and its ramifications for scheduling theory started to percolate through the research community. Suddenly, the inherent computational limits of most deterministic scheduling models became abundantly clear, explaining the resistance of some models to efficient solution. This gave rise to several responses, not necessarily mutually exclusive. Some researchers focused on identifying NP complete models and defining the borderline between the models that are NP complete in the ordinary and the strong sense (an important distinction we discuss presently). For an impressively long list of the earlier contributions of this genre, see Garey and Johnson (1979), which remains the classic reference on the subject. Studying complexity theory at more depth, as these researchers did, also yielded methods that were guaranteed to yield nearly optimal results in polynomial time for some models. Others focused more attention on devising efficient heuristics, and indeed it became easier to publish heuristic methods. Still others saw this as an opportunity to move to new pastures, of which, in our context, stochastic scheduling was an important one.

A more recent area of work that emerged after the discovery of NP-completeness addresses objectives with earliness/tardiness (E/T) penalties. The E/T objective is not a regular measure, which does make a significant difference even within the confines of deterministic scheduling. They are important because in practice finishing jobs too early may be costly as it leads to various forms of inventory holding costs. Indeed, the advent of these models was associated with the JIT (just-in-time) production control method, which penalized earliness as well as tardiness and strongly discouraged excessive inventories. But the field of stochastic scheduling was shaped just before the advent of these E/T models, at a period when deterministic models were all regular. As a result, very naturally and without any explicit deliberation, stochastic scheduling addressed itself to stochastic counterparts of deterministic models with regular performance measures. Such counterparts are defined by replacing important job descriptors from deterministic scalars to stochastic distributions, or possibly considering random machine downtime, and by aiming to optimize a traditional deterministic performance measurement by expectation. But what was missing for a long time thereafter was a focus on including safety time in schedules. That is a relatively new development that we call *safe scheduling*, and this is the first textbook that discusses such models. As it happens, the most studied safe scheduling models to date are stochastic counterparts of deterministic E/T models. In other words, there are two traditional limits that must be removed to achieve safe scheduling models: deterministic data and regular objectives. Conventional stochastic scheduling only addresses the former, whereas safe scheduling adds the latter. We discuss basic stochastic models in Chapter 6 and safe scheduling starting in Chapter 7.

The historical significance of computational complexity theory highlights its centrality in modern scheduling research, so we now present it in more detail than we did in the chapter, but we still refer the readers to Garey and Johnson (1979) for a sufficiently thorough introduction. The theory of NP-completeness involves *decision problems*, whereas in this text we are concerned with *optimization problems*. A decision problem is answered by either *yes* or *no*, whereas an optimization problem seeks some best value. For example, consider Figure 1.1 and suppose each interval in the horizontal scale represents 10 time units. We might focus on the makespan and compose decision problems as follows: Can all jobs be completed in at most 100 time units? 140 time units? 135? By the evidence in the figure, the answer to the first question is *no* because the work content the three resources must complete clearly exceeds 300 time units, so even if we could have all of them operate in parallel without idling it would take more than 100 time units to complete the schedule. The answer to the second question is *yes*, as the figure demonstrates a solution that completes in about 137 time units. However, because it is not clear by the figure whether another sequence might exist that would take 135 or less, it may be difficult to answer the third question. These instances demonstrate that we can approach an optimization problem such as identifying the earliest possible completion time by a series of decision problems. Thus, decision problems are not irrelevant to optimization, as we elaborate later. They also demonstrate that the complexity of an instance may be a function of the precise parameters. In response, complexity theory concerns itself with the worst-case scenario. For instance, the fact that we could answer the question in our example for 100 and for 140 with ease does not make the problem inherently easy. To continue, for any decision problem, an algorithm is a structured set of programmed actions that lead to a *yes* or *no* answer, as required. An algorithm with a worst-case running time bounded from above by a polynomial function of the input size required to describe an instance of the model is called *polynomial*. Problems that can be solved by polynomial algorithms compose an equivalence class called P.

Most scheduling problems, even deterministic ones, have no known polynomial solutions, however, so it is highly doubtful that they belong to P. These models are combinatorial, which means that their complexity is associated with the fact that there are many possible sequences that need to be implicitly or explicitly considered. There are exponentially many such sequences and we should not be surprised that identifying the best one is often difficult. However, suppose that a sequence were given (e.g., by a guess); then it would be easy to calculate its performance in polynomial time. The set of all problems where the performance measure of a given solution can be computed in polynomial time (and by comparing it to some target providing a *yes* or *no* answer to a decision problem) is a class defined as NP. In particular, all problems that can be solved in polynomial time satisfy this definition, and therefore P must be included in NP. What is less clear is whether there exist any problems that are in NP but not in P. The prevailing conjecture is that such problems do exist; i.e., that the set NP − P is not empty. We say that problem A is *transformed* (or *reduced*) to problem B if any instance of A can be formulated as an instance of B. If this transformation takes polynomial time, then the complexity status of problem A cannot exceed that of problem B. If we can also transform B to A in polynomial time, the two problems are computationally equivalent. Polynomial transformation is transitive; i.e., if A can be transformed to B and B can be transformed to C then A can be transformed to C (by simply following through with both transformations). The set NP-complete is an equivalence set of such problems, each of which can be transformed to all the others in polynomial time. Initially, for one problem in this set—called SATISFIABILITY—it was shown by Cook (1971) that *all* problems in NP can be transformed to it in polynomial time. This includes easy and hard problems in NP, ranging from polynomial ones to those suspected of being intractable, so SATISFIABILITY must be as hard as any problem in NP. This led to the definition of the class NP-complete as the class of all problems in NP to which all other problems in NP can be transformed in polynomial time. However, because all problems in NP can be transformed to SATISFIABILITY in polynomial time, any problem in NP to which SATISFIABILITY itself can be transformed in polynomial time must also be NP-complete (because polynomial transformation is transitive). Indeed Cook showed that SATISFIABILITY was not the only such problem. Notably, however, no NP-complete problem was ever transformed in polynomial time to a problem in P (or the conjecture that $P \neq NP$ would be disproved). Another way to show that a problem in NP is NP-complete is by *restricting* it to another NP-complete problem. Restricting problem A to problem B involves showing that problem B is a special case of problem A. (Restriction is the reverse of transformation: transforming A to B implies that the ability to solve B is sufficient for the solution of A. Restricting A to B implies that the ability to solve A is sufficient for solving B.) Using such methods, Karp (1972) listed 20 NP-complete problems. Garey and Johnson (1979) were able to list hundreds. By now there must be many thousands.

NP-complete problems, although they are equivalent by definition, still come in two categories that are very different in practical complexity, and often (but not always) we know which category a problem belongs to. One category, easier to solve, is the subclass of NP-complete problems that have *pseudopolynomial* solutions. These problems are often tractable in practice in spite of their NP-completeness. To explain the meaning of a pseudopolynomial solution, a simplified example may be useful. Suppose that our problem concerns n identical jobs and yet the solution takes O(n) time. One might think that this result is polynomial, and indeed it is polynomial in n, but it is *not* polynomial relative to the size of the necessary input because all it takes to describe n identical jobs is a string of length O(log n), and n is not a polynomial function of log n; instead, n is $O(2^{\log n})$, which is exponential in the input size, log n. However, if we actually have to process these n jobs, then it stands to reason that O(n) is not prohibitive in a practical sense even though the problem may be NP-hard in the mathematical sense. There are several other common terms by which problems that have pseudopolynomial solutions are known, including: NP-complete *in the ordinary sense*, NP-Complete *in the weak sense*, and NP-complete in the *binary sense*. The more difficult cases are described as NP-complete *in the strict sense*, *in the strong sense*, or in the *unary sense*, respectively. It may be worthwhile to elaborate on the terms *binary* and *unary* in this context. The essence of a pseudopolynomial case is that if the size of the input matched the size of the [integer] numbers that are involved, the problem would have a polynomial solution. But because in computers we can use binary inputs, which require O(log n) to describe a number of size n, the solution is not strictly polynomial in input size. The same would remain true with decimal inputs or any other base greater than or equal to 2. But suppose that we did not use the binary code but instead we used the unary code which represents the number n by n repetitions of the digit 1 next to each other. With unary coding, the size of the input would be such that a pseudopolynomial solution for binary coding would become polynomial. Thus, to say that a problem is NP-complete in the unary sense is the stronger statement: it remains NP-complete even if we give up the compactness of the binary coding. In such a case the problem can be complex even if its input requires only few small numbers.

In the text we will often mention models that are associated with NP-complete decision problems, and yet they can be solved for many jobs, say hundreds of jobs on a single machine. Typically, these are NP-complete in the ordinary sense. On the one hand, the existence of a pseudopolynomial solution does not imply that it is the best solution in practice. But on the other hand, there is a correlation and such problems are often easier to solve even with algorithms that are not pseudopolynomial. Furthermore, if a problem has a pseudopolynomial solution, it is by definition easier to solve for small inputs. But it is sometimes possible to reduce the size of any numerical input by the simple expedient of rounding numbers with many digits to fewer digits. As a result, it may be possible to solve pseudopolynomial problems at least approximately within any given time frame. By this we obtain a nearly-optimal solution for an NP-complete problem in polynomial time.

We now discuss NP-hard problems, described by Garey and Johnson (1979) as "at least as hard as the NP-complete problems." Liberally interpreted, this implies that any problem that can be shown to be at least as hard as an NP-complete problem can be called "NP-hard." Two typical examples of NP-hard problems are optimization problems associated with NP-complete decision problems and problems that can be restricted to NP-complete problems but cannot be shown to be in NP. For an example of the former, suppose that we can find the shortest makespan for the problem depicted in Figure 1.1, and denote that value by C. Then clearly we can answer any decision problem of the type "is there a sequence with makespan not exceeding *B*" for any *B* by simply comparing *B* to *C*. Thus the optimization problem (to find *C*) cannot be easier than the decision problem and if the decision problem were NP-complete the optimization problem is thus proved to be NP-hard. As an example of the latter, in our context, stochastic scheduling models often involve intractable distribution calculations (which we address by simulation) and continuous variables. As such, even if posed as decision problems they are not in NP (solutions cannot be verified in polynomial time). Nonetheless, they can be *restricted* to problems in that domain by modeling deterministic processing times as degenerate random variables with integer values. Because the general problem cannot be easier than a restricted instance, if the decision problem associated with a restricted version of such a stochastic problem is NP-complete we know with certainty that our original problem is at least as hard (intractable) as an NP-complete problem, and is thus NP-hard.

One might think that optimization problems are much more difficult than decision problems, but that is not necessarily the case. Optimization problems with integer inputs and strictly positive performance measures can be expressed as a series of decision problems by conducting a numerical search for the optimal value as follows:

1. Select any sequence and calculate its [objective function] value, say *U*. We now know that the answer to the decision problem "is there a sequence with value *U* or less" is "yes."

2. Let $L = \lceil U / 2 \rceil$ (where $\lceil x \rceil$ is the smallest integer not smaller than *x*; i.e., $\lceil x \rceil \geq x$ but $\lceil x \rceil - 1 < x$). If $L = U$, stop: *U* is the optimal value. Else, answer the decision problem "is there a sequence with value $\leq L$?" If "yes," let $U = L$ and repeat; else, continue.

3. [When we reach this step we have two distinct values *L* and *U*, of which *U* is feasible—the decision problem yields "yes"—and *L* is not. We maintain this structure from here on out but we reduce *U* or increase *L* until the unique optimal value is identified. This occurs when $L + 1 = U$.] Let $k = \lfloor (L + U) / 2 \rfloor$ (where $\lfloor x \rfloor$ is the largest integer not larger than *x*). If $k = L$, stop: the optimal value is *U*; otherwise, continue.

4. Answer the decision question for the value *k*. If the answer is "yes," set $U = k$; else, set $L = k$. Return to 3.

This procedure essentially halves the gap between *U* and *L* every time, and therefore it takes O(log(*U*)) time. Since in typical cases O(log(*U*)) is polynomial in the problem input size, the complexity of the optimization problem matches that of the decision problem.

From the analysis above, we can conclude that if a decision problem has a polynomial (pseudopolynomial) solution, then the associated optimization problem also has a polynomial (pseudopolynomial) solution. Thus, there is no complexity difference between decision problems and optimization problems. Under the assumption that NP ≠ P, operationally, all our optimization problems belong to one of three classes: NP-hard, open, and polynomial. An open problem cannot have a known polynomial solution.

Furthermore, the NP-hard group includes a “pseudopolynomial” subset. This does not imply, however, that we necessarily know exactly which category all problems belong to.

References

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

Cook, S. (1971) "The Complexity of theorem-proving procedures,” *Conference Record of Third Annual ACM Symposium on Theory of Computing*, 151–158.

Garey, M.R. and D.S. Johnson (1979) *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, New York.

Jackson, J.R. (1957) Networks of Waiting Lines, *Operations Research* 5, 518-521.

Johnson, S.M. (1954) "Optimal Two-and Three-Stage Production Schedules with Setup Times Included," *Naval Research Logistics Quarterly* 1, 61-68.

Karp, R.M. (1972) "Reducibility among combinatorial problems." In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103, Plenum Press, New York.

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.

Raiffa, H. and R. Schlaifer (1961) *Applied Statistical Decision Theory*, Harvard University Press, Cambridge, MA.

Rothkopf, M.H. (1966) "Scheduling with Random Service Times," *Management Science* 12, 707-713.

Smith, W.E. (1956) "Various Optimizers for Single Stage Production," *Naval Research Logistics Quarterly* 3, 59-66.

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