Research Notes for Chapter 9*

After discussing sources for the results in Chapter 9 and providing the proof of Theorem 9.5, we list some additional published results, including classical stochastic models. Then we provide preliminary analysis for safe scheduling in parallel machines. There are precious few, if any, published results in this area, so our purpose here is, again, to encourage further research.

Sources and Comments

Algorithm 9.1 is due to McNaughton (1959). Theorems 9.1 and 9.3 are due to Graham (1969), although the details of our proofs are different. Theorem 9.2 follows a template presented by Portougal (1993). The details of an efficient implementation of the Multifit algorithm are described by Coffman, Garey and Johnson (1978). Subsequently, error bounds for the Multifit algorithm have been refined, although its worst-case performance has not yet been discovered. The result given in the chapter, that $M/M^* \leq 72/61$, is due to Friesen and Langston (1986). An overview of further results is given by Kao and Elsayed (1990), but this continues to be an active research area and readers can easily find newer results. The idea to start off with an LPT application before implementing Multifit was tested by Lee and Massey (1988). They do not develop an error bound for this combination, but they provide an empirical study that suggests the size of the improvements possible. They also observe that the combined algorithm requires much less searching than Multifit alone. Although the proof of Theorem 9.2 does not extend to the Multifit algorithm, the combination of Multifit and LPT must still be asymptotically optimal—as we stated in the chapter—for the simple reason that it can only improve upon an asymptotically optimal solution.

Lee (1991) addresses a variation of the parallel machine model where the machines are not all available at time zero. Instead, machine i has a release date, $r_i \geq 0$, and is unavailable until that moment (see Exercise 9.5). This model would be appropriate in situations where jobs are released to the shop periodically (for example, every morning) and where, for each machine, the jobs of one batch must be cleared before a new batch can be started. It is still possible to use list scheduling in this environment, and Theorem 9.2 holds for any such list. Lee shows that LPT list scheduling yields an error bound of $3/2 - 1/2m$. The following table compares this bound with the error bound in the basic model, as given by Theorem 9.3.

---

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

Citation details: Baker, K.R. and D. Trietsch (2019) Research Notes for Chapter 9 in *Principles of Sequencing and Scheduling* (Wiley, 2019). URL: http://faculty.tuck.dartmouth.edu/principles-sequencing-scheduling/.

Machines m	2	3	4	5	10	20
$r_i = 0$	1.17	1.22	1.25	1.27	1.30	1.32
$r_i \geq 0$	1.25	1.33	1.38	1.40	1.45	1.48

Lee goes on to show that the error bound can be reduced to 4/3 by exploiting information about the r_i values. See Chang and Hwang (1999) for slightly improved bounds. Lee, He and Tang (2000) point out that Lee's original results rely on the assumption that all machines participate in the processing. Without that assumption, better results can be obtained. Better results can also be obtained for two machines (He, 1999).

Theorems 9.1 and 9.3 provide error bounds for increasingly more detailed heuristic procedures for identical parallel machines and unrelated jobs. When we consider uniform machines, the performance ratio of 19/12 for LPT list scheduling that we cited in the chapter is due to Dobson (1984). Even naive dispatching (assigning the next job on the list to the first available machine, regardless of speed) is still asymptotically optimal, as long as machine speeds are finite. The proof of Theorem 9.2 fails only for the more general case of *unrelated* machines, where each machine processes each job at a different speed. Algorithm 9.2 is due to Hu (1961). Algorithm 9.3 is due to Coffman and Graham (1972). They also discuss the potential advantage of preemption in that case, as presented in the chapter. Recall that although Algorithm 9.3 generalizes the longest-path notion of Algorithm 9.2 to arbitrary precedence relations, this generalization provides optimal schedules only for two machines and only for unit-length jobs. Beyond two machines, no further generalization seems possible, even with unit-length jobs. With regard to the number of machines, the makespan problem is NP-hard for $m \geq 3$, even for sets of unit-length jobs, and the corresponding worst-case bound is given by Theorem 9.4. See Lam and Sethi (1977) for details. The application of SPT for flowtime minimization appears in Conway, Maxwell and Miller (1967). They also show how to apply SPT to uniform machines. Horowitz and Sahni (1976) render the algorithm in detail and provide a proof of optimality. The seminal results about the use of SWPT dispatching are due to Eastman, Even and Isaacs (1964). We now repeat their lower bound and—within our proof of Theorem 9.5—an upper bound, which is also due to them.

Theorem 9.5: A Proof

Recall from the chapter that the experiment by Baker and Merten (1973) showed that the SWPT heuristic is effective and especially so for large n, which inspired Theorem 9.5. Recall also that we defined

$B(1) =$ the minimal value of F_W for the given job set if there were only one machine (obtained via SWPT), and

$B(n) =$ the minimal value of F_W for the given job set if there were n machines (obtained by assigning each job to a different machine)

leading to the lower bound for m machines ($1 \leq m \leq n$)

$$B(m) = \frac{1}{2m}\,[(m-1)B(n) + 2B(1)] \tag{9.4}$$

Theorem 9.5. When processing times and weights are sampled from distributions with finite variances, Algorithm H_1 with R = SWPT is asymptotically optimal with probability 1.

Proof.

»» Without loss of generality, we assume that all processing times are strictly positive. Otherwise, jobs with zero processing times would be scheduled first and contribute zero to the objective function value. Because $B(n) = \sum p_j w_j$, (9.4) can be rewritten $B(m) = B(1)/m + 0.5(m-1)\sum p_j w_j/m$. Eastman, Even, and Isaacs (1964) also provided an upper bound for H_1 with R sequenced by SWPT, namely $B(1)/m + (m-1)\sum p_j w_j/m$. To prove the theorem it is sufficient to show that when the gap between the bounds is divided by a lower bound (including any value $B \le B(m)$), the result tends to zero as n tends to infinity; i.e., as $n \to \infty$, $[0.5(m-1)\sum p_j w_j/m] \,/\, B(m) \to 0$ (*w.p.*1). We show that in terms of expected values. Convergence *w.p.*1 then follows by the Law of Large Numbers.

Let $p_{0.5}$ be the median processing time, similarly let $w_{0.5}$ be the median weight, and let $\lambda = \max\{\mathrm{E}(p \mid p > p_{0.5})/\mathrm{E}(p \mid p < p_{0.5}), \mathrm{E}(w \mid w > w_{0.5})/\mathrm{E}(w \mid w < w_{0.5})\}$. Because all processing times are strictly positive, λ is finite. By (9.4) we see that $B(m) > B(1)/m$. But m is fixed, so it is sufficient to show that as $n \to \infty$, $\mathrm{E}[\sum p_j w_j]/\mathrm{E}[B(1)] \to 0$, or, equivalently, that $\mathrm{E}[B(n)]/\mathrm{E}[B(1)] \to 0$. Because processing times are sampled independently from a distribution with a finite variance, for any given sample of n processing times, when we sample job $(n+1)$, $\mathrm{E}[p_{(n+1)}]/\sum_{j=1,\ldots,n} p_j = 1/n$. Similarly, for any $\varepsilon > 0$ (but as small as we may wish) there exists a value $n_0 \ge \lceil 2\lambda/\varepsilon \rceil$ such that for any $n \ge n_0$, $\mathrm{E}[p_{(n+1)}]/\sum_{j=1,\ldots,n} \delta(p_j \le p_{0.5}) p_j < \varepsilon$ (i.e., by the $\delta(p_j \le p_{0.5})$ function we select only the smaller 50% of the sample). An analogous result holds for weights. Let $B_0(1)$ be the minimal weighted flowtime of the first n_0 jobs on a single machine, and let $B_0(n_0) = \sum p_j w_j$ where the summation is for $j = 1, \ldots, n_0$. For any $n > n_0$, suppose we add job $(n+1)$ at the correct position in the SWPT sequence of the first n jobs. Without loss of generality, assume it should be placed between jobs $[k]$ and $[k+1]$, but continue to identify the other jobs by their former positions; so the sequence becomes [1], [2], …, $[k]$, $(n+1)$, $[k+1]$, …, $[n-1]$, $[n]$. Job $(n+1)$ contributes $p_{(n+1)}w_{(n+1)}$ to $B(n+1)$ and we now check how much it contributes to $B(1)$. We distinguish two cases. In Case 1, $k \ge n/2$, and we focus on the flowtime of the inserted item. In case 2, $k < n/2$, and we focus on the additional flowtime imposed on the subsequent items from the inserted item onwards. Either way, the result is a strict lower bound on the additional flowtime that the insertion entails. Consider case 1 first. The flowtime of jobs [1] through $[k]$ is not changed. The flowtime of job $(n+1)$ itself is given by $p_{(n+1)} + \sum_{j=1,\ldots,k} p_{[j]}$, and it should be multiplied by $w_{(n+1)}$. In case 2, jobs $[k+1]$, …, $[n-1]$ all have $p_{(n+1)}$ added to their flowtimes, thus adding $p_{(n+1)} \sum_{j=k+1,\ldots,n} w_{[j]}$ to $B(1)$. In case 1, after dividing the expected increase of both $B(n+1)$ and $B(1)$ by $p_{(n+1)}$, we see that the former increases by less than ε times the increase of the latter. In case 2, the same observation applies after

dividing by $w_{(n+1)}$. Therefore, for any $n > n_0$, each unit added to $B_0(n_0)$ implies at least $1/\varepsilon$ ($> n_0$) units added to $B_0(1)$. In the limit, as $n \to \infty$, $E[\sum p_j w_j]/E[B(1)] < \varepsilon$, but ε is as small as we wish so the limit is 0. ««

Unrelated Machines

Consider the C_{max} problem with parallel unrelated machines, denoted $Rm \mid\mid C_{max}$.* It is straightforward to formulate the problem as an integer program. To that end, define an indicator variable x_{ij} as 1 if job j is assigned to machine i, or 0 otherwise. The total load on machine i is thus $\sum_j p_{ij}x_{ij}$ and $C_{max} \geq \sum_j p_{ij}x_{ij}$. The task is to minimize C_{max} subject to m such constraints (one for each machine). Additional constraints of the form $\sum_i x_{ij} = 1$ are used for each job to assure that it will be assigned exactly once. It follows that small instances can be readily solved by generic IP solvers. Nonetheless, the problem is NP-hard in the strong sense and more tailored approaches and heuristics are required for large instances. Horowitz and Sahni (1976) present both optimal and ε-approximation algorithms. They do not provide computational results, however. Ibarra and Kim (1977) present heuristics with worst case error bounds of $O(m)$. Davis and Jaffe (1981) present heuristics with $O(m^{0.5})$ worst case error bounds, again without computational results. Van de Velde (1993) presents and tests a new optimization algorithm based on Lagrangian relaxation, and a new heuristic. The former solves to optimality some instances with up to 200 jobs and 20 machines, but not all combinations can be solved without violating a bound of 100,000 nodes in the search tree. The results indicate that average computation time increases with the number of machines more so than with the number of jobs, but large variation exists among problems of the same size (which is the rule and not the exception with implicit enumeration algorithms). Reportedly, good exact solutions were obtained by Martello, Soumis and Toth (1997) and by Mokotoff and Chrétienne (2002), both utilizing strong cuts. Ghirardi and Potts (2005) apply a version of beam search that allows revisiting previous decisions by re-sequencing existing branches in the search tree. They also discuss earlier contributions in detail. Empirical results concerning good heuristics for this problem indicate that they may be asymptotically optimal. For instance, Potts (1985) observes that if we solve the LP relaxation of the problem, then at most $m - 1$ jobs have to be split to fractional parts. A heuristic solution is to keep the allocation of the $n - m + 1$ other jobs as a partial schedule and assign the fractional jobs optimally. The heuristic is polynomial for any given m (because we can schedule the $m - 1$ jobs by complete enumeration in constant time), but it becomes exponential when m is not fixed. Whereas the worst-case error bound of this heuristic is 2, in practice it is much better. Indeed, it is immediate to show that this heuristic is asymptotically optimal, as $n/m \to \infty$, if jobs follow reasonable regularity conditions. It is also easy to show that scheduling the fractional jobs by a secondary heuristic can overcome the exponential complexity when m is not given, and yet retain asymptotic optimality: for that purpose, it does not even matter how these jobs are allocated. An open research question is to prove asymptotic optimality for other such heuristics and to compare their convergence rates. Finally, the observation that at most $m - 1$ jobs have to be split into fractional parts in the LP relaxation provides another perspective on the role of m in the computational difficulty of the problem. One can construct a branch and bound procedure for the problem, branching on such fractional variables. Initially, there would be up to $m - 1$ nodes on which to branch, but each such node

* Here, R is the notation for unrelated parallel machines whereas uniform machines would be denoted by Q.

could lead to m potential offspring, and so on. Hence, for large m, even a depth-first branching designed to obtain an initial upper bound could take significantly longer than with small m.

The F_w-problem with unrelated machines, $Rm \mid\mid \sum w_j C_j$ has also received a fair amount of attention. The problem is strongly NP-hard unless m is fixed, in which case it remains NP-hard but only in the ordinary sense. When all weights are equal, however, there is an easy polynomial solution by a reduction to an LP assignment problem or to a related network flow problem. The first such solution was given by Horn (1973). Horn loads machines backwards and observes that if job j is allocated to the kth position (from the end) on machine i, it contributes kp_{ij} to the total flowtime downstream. That value is taken as the cost of allocating job j to machine i at that position. The waiting time part of the flowtime of the job is accounted for as part of the total downstream flowtime contribution of upstream jobs, if any. What remains is to match the n jobs to the nm possible positions so as to minimize the total cost. To see that this problem is identical to an LP assignment problem, we can introduce $n(m-1)$ dummy jobs, with zero costs, to fill the empty positions in an nm by nm assignment tableau. Alternatively, we can solve the problem as a network flow model. Both models have well-known polynomial time algorithms. The weighted case is more complex: recall that even for identical machines it is already NP-hard. Nonetheless, more than one set of authors discovered independently that tight bounds can be obtained by a mathematical programming approach using column generation (van den Akker, Hoogeveen, and Van De Velde, 1999; Chen and Powell, 1999), thus enabling optimal solutions for moderate-sized instances. That approach is applicable to identical, uniform or unrelated machines. As discussed by van den Akker, Hoogeveen and Van De Velde, it also applies to similar problems where, after allocating the jobs to machines, the sequence on each machine is easy to obtain. An example is the weighted number of tardy jobs problem where only on-time jobs must be allocated, and given such an allocation the schedule of the jobs selected to be on time follows EDD.

Stochastic Counterpart Parallel Machine Models

The observation that heuristics are sometimes more robust when applied to stochastic models is due to Weiss (1992). That seems to occur often when the heuristic is asymptotically optimal in the deterministic case. One such instance is the m-machine C_{max}-problem with exponentially distributed jobs. As stated in the chapter, LEPT dispatching is not only optimal but also yields the stochastically minimal makespan. That feature follows from a more general result given by Weber (1982). Weber also lists earlier sources that showed that LEPT solves the stochastic counterpart (that is, that it yields the minimal expected makespan). By contrast, SEPT minimizes total flowtime for the exponential distribution. Thus an inherent conflict appears to exist between the two main objectives, makespan and flowtime. In addition to the paper by Weber (1982), relevant results and earlier sources also appear in Weber, Varaiya and Warland (1986) and in Pinedo (2002).

On the Variance Ratio and the Value of Information

In Example 9.7 we showed that although LEPT dispatching achieves an expected makespan larger than the deterministic counterpart, it does provide a 10% savings relative to assigning jobs to machines at the outset (from 7.004 to 6.271). This result illustrates an

important point: in the stochastic case, the use of dispatching implicitly involves collecting useful information during the process. By following LEPT, we implicitly collect information about the processing times of the jobs with the highest processing time variances, because in the exponential case those are also the jobs with the highest mean processing times. To generalize this insight beyond the exponential distribution, let X_0 be a nonnegative random variable with $\mu_0 = 1$, a finite variance and a given cdf, $H_0(x)$. Suppose that all job processing times have distributions with the same shape as that of X_0 in the following sense: if p_j has mean μ_j and distribution $H_j(x)$, then $H_j(x) = H_0(x/\mu_j)$. That is, the processing time cdfs are essentially identical except for scaling by their means. In such a case, all coefficients of variation are equal to a single constant, *cv*. Furthermore, the processing times are stochastically ordered. When all processing times are exponential, they fit this template (with $cv = 1$). Lognormal processing times with constant *cv* fit the template as well.

By *variance ratio* we refer to the ratio of variance to the mean. If we calculate the variance ratios of these processing times, we obtain

$$\sigma_j^2/\mu_j = cv^2\mu_j$$

Because *cv* is a constant, the variance ratios are ordered by the means. Thus, for the exponential case and any other constant *cv* case, SEPT is equivalent to sorting by smallest variance ratio (SVR), whereas LEPT is equivalent to sorting by largest variance ratio (LVR). We also expect random variables with high coefficients of variation to have high variance ratios as compared to jobs with the same mean but smaller coefficients of variation. Broadly speaking, we can say that we gain more information from completed jobs with higher variance ratios. But if each job has its own coefficient of variation, cv_j, then the variance ratio is $cv_j^2\mu_j$. Therefore, by LVR sequencing with non-similar jobs (whose distributions are not scaled copies of each other), we favor jobs with high coefficients of variation and high means. By completing a job, we gain perfect information about its processing time, whereas unprocessed jobs are still subject to variance. By sequencing LVR, we resolve the maximal amount of variance per unit of expected processing time in a greedy manner. We effectively postpone the jobs that are closest to deterministic so we can utilize the machines better towards the end of the schedule. In contrast, if we were to leave high variance jobs to the end, we would risk incurring a large gap between the completion times of the first machine and the last machine.

Intuitively, LEPT would likely be most attractive when coefficients of variation are low, whereas LVR becomes attractive when coefficients of variation are high. In addition, the amount of information available with dispatching is also a function of the coefficient of variation. Thus, the ability to implement dispatching (which implies the ability to change tentative sequencing decisions quickly), is especially important in a stochastic environment because it facilitates better use of information. For instances where LEPT and LVR are not identical, a heuristic approach is to sort by $\max\{\mu_j, \sigma_j^2/\mu_j\}$. The idea here is to schedule both very long jobs and highly variable jobs early, but $\sigma_j^2/\mu_j = cv^2\mu_j$ so $\max\{\mu_j, \sigma_j^2/\mu_j\}$ is achieved by some σ_j^2/μ_j only if that job has $cv_j^2 > 1$. (Recall that for exponential variables $cv^2 = 1$.) Jobs with $cv_j^2 > 1$, however, tend to have decreasing completion rates, so if they don't finish early they are likely to take more than μ_j. Another heuristic is to dispatch by $\lambda\mu_j + (1 - \lambda)(\sigma_j^2/\mu_j)$, where $0 \le \lambda \le 1$. By simulating the results of various λ values (using a stored

sample) we can choose a good λ value. Similarly, but not identically (because the variance ratio is not a linear function of the mean and the variance), we could use the mean and the variance directly; i.e., $\lambda\mu_j + (1 - \lambda)\sigma_j^2$, where $0 \le \lambda \le 1$. The latter approach is identical to the one we presented in the Research Notes for Chapter 8 for the safe scheduling versions of the shortest route problem and the TSP. It remains an open question which of the two combinations (mean and variance ratio or mean and variance) is more effective. Theoretically, we could even use a more elaborate combination giving weight not only to mean and variance but also to the variance ratio directly: $\lambda_1\mu_j + \lambda_2\sigma_j^2 + (1 - \lambda_1 - \lambda_2)(\sigma_j^2/\mu_j)$.

Stochastic Load Balancing for Makespan Minimization and for Safe Scheduling

As we have seen, dispatching is effective for balancing loads across multiple machines. We observed that phenomenon for highly variable jobs and for deterministic processing times. In both cases, list scheduling heuristics, which are inherently suited for dispatching, provide good performance bounds and asymptotically optimal sequences. We now turn to non-dispatching solutions where we must allocate the jobs to machines in advance. This analysis is important in practice because dynamic dispatching is not always possible. For example, the machines may reflect separate plants that are far apart or jobs require preparations that cannot be easily changed. In chapter 18 we refer to such preparations as *implicit subprojects*. The need for such preparations is quite common. Sometimes the practical response is to freeze the schedule for a given period of time and allow changes only later. That approach implies a rolling horizon type of planning that is essentially equivalent to a dynamically updated *non-dispatching* schedule. Thus it is important to address non-dispatching schedules even if we recognize that we may change them later. Yet one more reason to use a static allocation is that by dynamic dispatching aimed at minimizing the makespan, we tend to increase (and often even maximize) the flowtime or the weighted flowtime. In the deterministic case, the remedy is to reverse the sequence on each machine after the allocation step, to SPT or to SWPT. Similarly, for any static allocation, we can use SEPT or SWEPT for each machine. But it is impossible to reverse a dispatch-based sequence and still maintain its efficient use of information. This conflict lies at the heart of the question whether we should use dynamic dispatching or schedule in advance. If we wish to utilize the information revealed during processing, then dynamic dispatching is imperative, but it is not cost free and not always possible.[*]

Although preemption may be prohibited in such models—especially when the reason we use a fixed schedule is the need to prepare in advance for every job—we may nonetheless allow preemption as a relaxation to obtain bounds. Suppose we could split each job into stochastically independent parts such that if we assign a fraction $0 < f < 1$ of job j to one part, then the expected processing time of this part is $f\mu_j$ and its variance is $f\sigma_j^2$. Calculating the variance ratio for this part we obtain $f\sigma_j^2/f\mu_j = \sigma_j^2/\mu_j$. That is, the variance ratio of the part equals that of the whole job (which implies that the coefficient of variation

[*] In Chapter 18 we discuss the weighted flowtime objective in a project environment. In that case, flowtime is measured from the start of an activity until the end of the project. In such an environment the conflict between makespan and flowtime disappears serendipitously. A similar serendipitous structure applies in our current context when all jobs are delivered to the customer together but flowtime is only measured from their release date (which in turn is a decision variable). One can view the delivery to the customer as a single project. To utilize dispatching in such an environment involves making release dates decisions dynamically for the next few jobs. These decisions depend on the status of the jobs currently in process.

is *not* the same). With this flexibility, a simple idea is to divide each job into m equal statistically independent parts, each with $1/m$ of the mean and $1/m$ of the variance, and allocate one part to each machine. Clearly, the distribution of the total processing time of each machine would be the same. This allocation would yield the most balanced allocation possible. One might hypothesize that such a perfectly balanced allocation would be optimal for makespan minimization as well as for minimizing safety time and expected E/T cost. Such a hypothesis would be only partly true, however. We first discuss to what extent that hypothesis is true and then we expose its weakness. We start with supportive evidence for the validity of this conjecture for the case of independent normal processing times with two machines. (The normality assumption is not restrictive in this case because the sum of independent random variables tends to be normal.) We show that balancing the load minimizes the expected makespan, which supports the hypothesis. Nonetheless, we also prove a counterintuitive result: for the purpose of minimizing the expected makespan, it does not matter how the variance is allocated between the two machines even though a Jensen gap is involved.

Theorem RN9.1. For jobs with independent normal processing time distributions processed on two machines with a makespan objective and no dispatching, the optimal deterministic counterpart job allocation is also optimal for the stochastic counterpart.

Proof.

»» For any given allocation of jobs to machines, let X denote the (normal) distribution of the convolution of the jobs assigned to machine 1, and let Y denote the convolution for the other jobs. Without loss of generality, assume $\mu_X \le \mu_Y$ and let $\Delta = \mu_Y - \mu_X$, $\sigma^2 = \sigma_X^2 + \sigma_Y^2$, and $z = -\Delta/\sigma$. Our task is to show that minimizing Δ is optimal for the stochastic counterpart makespan problem, regardless of the variance allocation. If $\sigma = 0$, the theorem is trivial; therefore, equivalently, we must show that maximizing z towards 0 is optimal. The expected makespan of the shop is given by $E(\max\{X, Y\}) = \mu_Y + E((X - Y)^+)$. But by symmetry with (B.16), $E(\max\{X, Y\}) = \mu_Y + \sigma[\varphi(z) + z\Phi(z)]$. Let μ_0 denote the total mean workload divided by 2 (that is, μ_Y when $\Delta = 0$), so we can write:

$$E(\max\{X, Y\}) = \mu_Y + \sigma[\varphi(z) + z\Phi(z)] = \mu_0 + \sigma[\varphi(z) - z[0.5 - \Phi(z)]] \;\; ; \;\; z \le 0$$

Because $z \le 0$, we have $0.5 \ge \Phi(z)$, so $-z[0.5 - \Phi(z)] \ge 0$. Taking the derivative by z, we obtain

$$\frac{d}{dz}[\max\{X, Y\}] = \sigma[\Phi(z) - 0.5]$$

For $z < 0$, $\Phi(z) < 0.5$ and the derivative is negative. Thus, increasing z towards 0 balances the load and reduces the expected makespan. ««

This particular proof cannot be generalized to more than two machines. It is also specific to the normal distribution, although it could be extended to any other distribution by the central limit theorem, as well as to jobs with linearly associated processing times. However, our analysis so far concerns the stochastic counterpart problem and does not imply that variance allocation is immaterial for safe scheduling purposes. Indeed, in safe scheduling, the whole distribution of the makespan counts, whereas Theorem RN9.1 concerns only the mean. Figure RN9.1 depicts the distribution of the makespan for four two-machine instances with an equal average load on each machine of 20 time units. The total variance on both machines is 17. That total variance is allocated to the two machines as per the next table, where we also list the resulting standard deviations and the ratios between those values.

	Case 1		Case 2		Case 3		Case 4	
	variance	stdev	variance	stdev	variance	stdev	variance	stdev
Machine 1	17	4.12	16	4	13.6	3.69	8.5	2.92
Machine 2	0	0	1	1	3.4	1.84	8.5	2.92
ratio	∞	∞	16	4	4	2	1	1

The figure was generated by numerical calculations performed by an Excel spreadsheet. These calculations utilize the equation $F_{max}(x) = F_1(x)F_2(x)$. The expected makespan in all cases is about 21.65. As the table indicates, in case 1, the variance is allocated fully to machine 1 whereas the other machine has deterministic processing time. This case has the longest tail to the right and a vertical segment at μ_0 (=20). The second case, depicted by a red line, involves a variance allocation in a ratio of 1:16 (i.e., standard deviations ratio of 1:4). The third case, depicted by a blue line, involves a variance allocation in a ratio of 1:4 (i.e., the standard deviations are 1:2), and the last case, in green, depicts a perfectly balanced variance allocation. Consider safe scheduling models with prescribed service levels. By the figure it is quite clear that for any service level target below 0.25 (the point where all cdfs intersect), the balanced case is best, between 0.25 and approximately 0.7, the unbalanced distribution is optimal (and it can be shown to be optimal at least between 0.25 and 0.5 in general), the perfectly balanced case is preferred again for higher service levels above 0.9 whereas the other two cases are approximately optimal between 0.7 & 0.8 and 0.8 & 0.9. (It is difficult to select the precise balance that is optimal at that range, but the differences are small.) The important point is that no single policy is universally optimal. The issue becomes even more complicated when we have the option to trade off mean and variance. We elaborate on these points below.

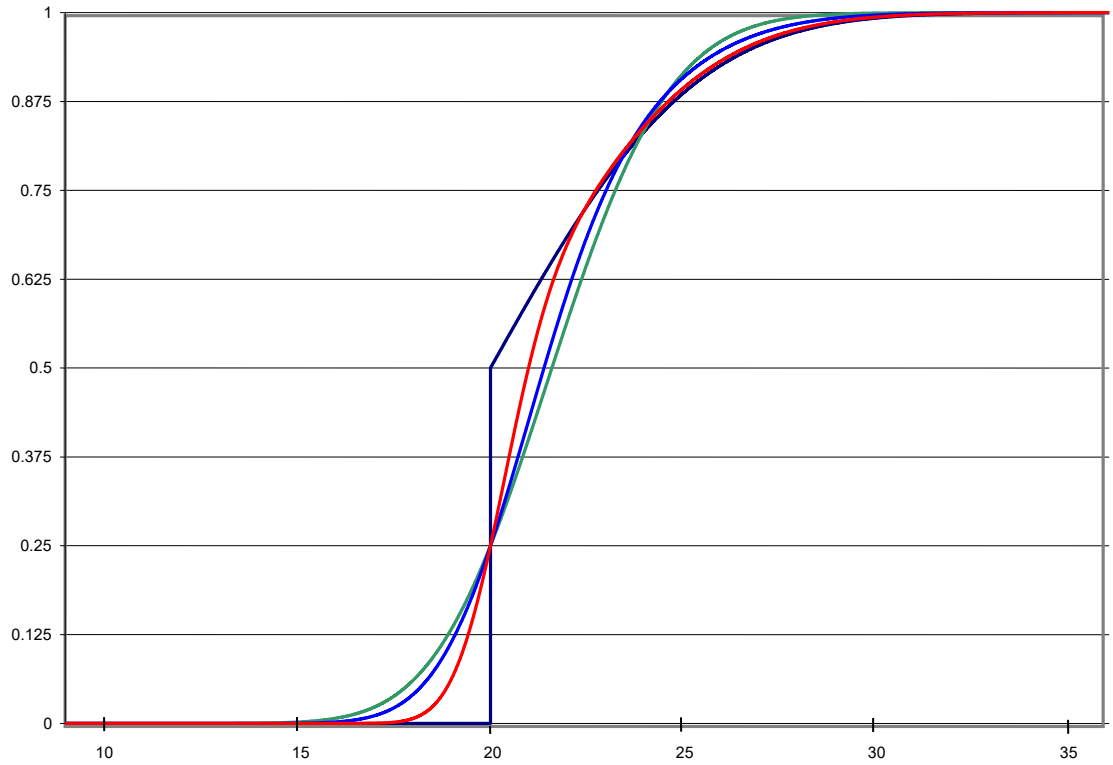

Figure RN9.1: The makespan cdf for balanced and unbalanced variance allocations

To study the tradeoff between mean and variance, we shift our focus to the problem of setting a due date so as to trade off tightness against expected tardiness. Assume that machine loads are not perfectly balanced and consider a case where the maximum processing time of some machine exceeds the due date. If a particular machine has both the maximum mean and maximum variance then it is more likely to be the one that is most tardy. Now consider the case where machine 1 has lower load but higher variance. In that case, because the difference between two normal variables is normal, the probability machine 1 completes last is less than 50%. However, if machine 1 is last, it may be more likely to also be tardy, and when tardiness is sufficiently large, it is more likely to be due to machine 1. The intuitive conclusion is that to some extent at least mean and variance can be traded off against each other. We show that this intuition is correct when the optimal service level exceeds 50%. To this end, assume that we have more jobs than machines ($n > m$) and that we are allowed to preempt at most $m - 1$ jobs. Assume further that we use some predetermined list and follow the pattern of Algorithm 9.1, but not necessarily such that the deterministic counterpart will be optimized: that is, we can specify different mean loads on each machine. (Instead of the one M^* value in the original algorithm, we now specify a different target for each machine such that the sum of the targets equals the total workload.) Define the *criticality* of machine i, denoted κ_i, as the probability that it will be the last one to complete its load and thus determine the makespan. Given any initial list for the algorithm to follow, we can specify the targets in such a way that given criticality objectives would be achieved (and it does not really matter if we know how to do so efficiently as long as such targets exist). In the perfectly balanced case, the criticalities of

all machines are equal, but so is the mean load on each machine. Under the current restriction, however, it may not be possible to achieve both equal loading and equal criticalities. If no single job dominates the makespan, however, it is possible to preempt jobs (i.e., set the load targets for the given initial sequence used by the algorithm) in such a way that the criticalities will be equal, whereas by setting equal load targets we can obtain equal loading (but not both at the same time). The question is which is more important.

Theorem RN9.2. Given a preliminary list, and subject to the use of Algorithm 9.1 with appropriate load targets to preempt at most $m - 1$ jobs, the expected makespan is minimized by loading the machines in such a way that all criticalities equal $1/m$.

Proof.

»» We prove the theorem for two machines, by contradiction. The proof can be extended to $m \geq 2$ machines by induction, but we omit the details. Assume a loading with unequal criticalities minimizes the makespan. Let M_i denote the makespan of the jobs on machine i and notice that the expected makespan is given by

$$\kappa_1 \mathrm{E}(M_1 \mid \text{machine 1 is critical}) + \kappa_2 \mathrm{E}(M_2 \mid \text{machine 2 is critical}).$$

In spite of the conditioning, if we transfer δ units of expected load from one machine to the other, the conditional expectations change by $-\delta$ and $+\delta$, respectively (although the probability of the relevant event increases with the load). Without loss of generality, suppose that $\kappa_1 > \kappa_2$, then it is possible to transfer a small enough load δ from the last job scheduled on machine 1 to the first job on machine 2 (i.e., change the fractions of that job allocated to each machine) so that κ_1 still exceeds 0.5. This transfer increases the expected load on machine 2 and reduces it on machine 1. As a result, with a probability > 0.5, we reduce $\mathrm{E}(M_1 \mid \text{machine 1 is critical})$ by δ and with the complementary probability, we increase $\mathrm{E}(M_2 \mid \text{machine 2 is critical})$ by δ. Hence, the expected makespan must decrease at least by $(\kappa_1 - \kappa_2)\delta > 0$ (where κ_1 and κ_2 are the new values), thus contradicting the assumption that the solution was optimal. ««

For two machines with normal distributions and equal loads the condition is indeed satisfied: the criticalities of the two machines are equal. A similar argument can be used to prove the following safe scheduling result:

Theorem RN9.3. Given a preliminary list, and subject to the use of Algorithm 9.1 (with appropriate load targets) to preempt at most $m - 1$ jobs, let the objective be to set a due date d and minimize

$$d + \gamma \mathrm{E}(T)$$

Then we should set the due date and balance the load on the machines in such a way that all machine criticalities equal $1/m\gamma$.

By setting $m = 1$ we see that this is a generalization of Theorem 7.2, and indeed the optimal service level is precisely the same, namely $(\gamma - 1)/\gamma$. But for $m \geq 2$ and $\gamma \geq 2$ (the minimal value for which the service level is 50% or higher), it follows that the due date should be set no earlier than the highest mean (or the machine with the highest mean will violate the due date more than 50% of the time and when the due date is violated, tardiness is implied). In this scenario, two equally-loaded machines with different variances will not satisfy the equilibrium condition of the theorem. Instead, the one with the higher variance will tend to contribute more than its share of maxima, and therefore it will tend to have excessive criticality. Thus we see that there is a tradeoff between mean and variance even for two machines with normal processing times when the required service level is at least 50%.

Theorems RN9.2 and RN9.3 provide necessary conditions for the respective objectives. The conditions are not sufficient because they do not guarantee the best allocation of jobs to machines (except for the jobs that are selected to be preempted, whose parts are indeed allocated optimally). But by selecting the best preliminary list in each case, we can try to improve the objective. For a sufficiently high service level, however, we may use the results above—such as the right tail of the balanced case in Figure RN9.1 and the observation that we need a service level above 50% to interpret Theorem RN9.3 as we did—to argue that the best list is the one that yields a solution that is as balanced as possible, not only in terms of the necessary condition (equal criticalities) but also in terms of balanced load. That is, we look for the most balanced loading that satisfies the theorems. Now, if we remove the relaxation and allow no preemption, the results suggest that we can measure the quality of an allocation by some measure of criticality balance as well as some measure of load balance. For example, we may use $\lambda(\kappa_{max} - \kappa_{min}) + (1 - \lambda)(\mu_{max} - \mu_{min})$, where $0 \leq \lambda \leq 1$ and μ_{max} (μ_{min}) is the mean load on the most (least) loaded machine. Here, $\lambda = 0$ corresponds to the deterministic counterpart solution, whereas $\lambda = 1$ addresses criticalities only. A more direct approach is to estimate the objective function value. This, in turn, can be done by employing a stored sample. It is clear, however, that optimizing the objective is not easier than the deterministic case, and for this reason we now return our attention to heuristics. (For convenience, we ignore the case in which one job dictates the makespan distribution, which is inherently easy.)

As we discussed already, LPT can be generalized in two "pure" ways: LEPT and LVR. We also introduced the idea to use the maximum of μ_j and σ_j^2/μ_j to represent job j. However, LVR is inherently associated with dynamic dispatching, whereas now we assume a static sequence. For this reason, our concern boils down to measuring the total mean load and the total variance on each machine, and hence there is no particular reason to use LVR. LEPT is more applicable, and may indeed produce a useful initial solution, but it is not sufficient because it ignores the variance completely. So we should look for another adaptation for LPT, if possible. Furthermore, we should also consider whether we can generalize Multifit. Recall that in the deterministic case, there is an advantage to starting the Multifit procedure with an upper bound obtained by LPT. Ideally, we should generalize both approaches for joint application. Unlike the deterministic case, however, we cannot

make the assumption that all processing times must be integers. That assumption is not crucial, as we can always approximate continuous numbers by integers. The challenge is to measure each job by a single scalar that is more sophisticated than the expected processing time, because both LPT and Multifit rely on the use of a single scalar for each job.

To make progress here it is useful to consider the lower bound associated with a perfectly balanced load. Furthermore, we assume that $1/m\gamma$ is sufficiently small to justify ignoring the possibility that two machines will exceed the due date at the same time (and only one is critical). Under this solution each machine has a mean $\mu_0 = \Sigma\mu_j/m$, and variance $\sigma_0^2 = \Sigma\ \sigma_j^2/m$. Given these values, the assumption, and γ, the optimal due date is associated with $SL = 1 - 1/m\gamma$, and thus we should set the due date to $d_0 = \mu_0 + k^*\sigma_0$, where $k^* = \Phi^{-1}(1 - 1/m\gamma) > 0$ (for $m \geq 2$ and $\gamma > 1$). If we also assume that we can load the machines with variance that is not far from σ_0^2, the marginal contribution of σ_j^2 to the standard deviation of the machine to which job j is allocated is $\sigma_j^2/2\sigma_0$. Using this value and Equation (B.17), we can measure the "length" of job j as $\mu_j + m\gamma\varphi(k^*)\sigma_j^2/2\sigma_0$. That is, we use an approximation of the correct tradeoff price for each unit of variance. Then, for each machine, we can identify the true mean load and variance allocated for it. Furthermore, for each machine, we can identify the due date associated with $SL = 1 - 1/m\gamma$ (although this is not precisely optimal for an unequal load). It is also possible to compare the results to those obtained by LEPT. At this stage we are ready to embark on a Multifit procedure designed specifically to minimize the maximal due date obtained on any machine by allocating jobs in decreasing "length" order. (It can be shown that the final optimal due date is bounded from above by this maximum.) In this application, let d_{UB} denote the current upper bound on the due date, obtained by the maximal due date in the best feasible solution; let d_{trial} denote a trial value for d, and let d_{LB} denote a lower bound (initially, set $d_{LB} = d_0$). To set a new trial value, at each stage we set $d_{trial} = (d_{LB} + d_{UB})/2$. Upon success, we set $d_{UB} = d_{trial}$, and upon failure we set $d_{LB} = d_{trial}$. We stop the procedure when we consider the bounds close enough, and the last upper bound is the final allocation. All that remains is to describe the first-fit-decreasing application in this context. To this end, we measure the load on a machine by the due date associated with the jobs already allocated to it (i.e., an unloaded machine has a due date of 0 and the most loaded machine has the highest due date). We test the fit of the "longest" unscheduled job by asking whether, after adding it to a machine, the updated due date of that machine exceeds d_{trial}, in which case the job has to be allocated to the next (less loaded) machine. Testing the performance of this approach is an open research challenge.

We conclude our analysis of the stochastic makespan problem with a very simple result that is, however, unpublished. The variance of the makespan cannot exceed the total variance of all jobs. (For a proof, see our Research Notes for Appendix A.)

Theorem RN9.4. Given any static job allocation to m parallel machines, the variance of the makespan is bounded from above by the sum of marginal variances of all machines.

Here, the marginal variance of a machine is the variance of the completion time of the last job allocated to that machine. We do not require statistical independence. However, when jobs are independent, the marginal variance of each machine equals the sum of variances of the jobs allocated to it and the sum of all m marginal variances is thus equal to the sum of the variances of all jobs.

An important corollary of this theorem is that the variance of the Parkinson distribution cannot exceed the variance of the internal random variable (because the variance of the constant is zero). From the proof, it is also clear that if the probability of hidden earliness is positive, the relationship is strict. As we note in Appendix A, however, this reduction of the variance comes at a cost of an increase in the mean.

Minimizing Total Flowtime

As we saw, LEPT and LVR are conducive to balancing the load on machines and thus reducing the makespan. However, as in the deterministic counterpart, if we wish to minimize total flowtime without preemption, LEPT is the opposite of what we should do. Instead, we should use SEPT (Bruno, Downey and Frederickson, 1981). Furthermore, with ICR distributions, SEPT is optimal even for the preemptive case (without actually resorting to preemption) and the exponential distribution satisfies this condition. The optimality of SEPT holds because, for an ICR processing time, a job selected for its small expected processing time becomes even more attractive after receiving some processing because the expected time to complete it decreases. Weber, Varaiya and Warland (1986) showed that a similar result holds when processing times are stochastically ordered (even if they are not ICR), and again the exponential distribution satisfies the condition. Similar results have also been shown to hold with intree precedence relationships, provided that processing times are stochastically ordered. As a rule, SEPT is at the very least an excellent heuristic wherever SPT is good in the deterministic case. With decreasing completion rates, however, preempting jobs after some processing, when their remaining expected processing time has increased, may be conducive to flowtime reduction.

Modeling with Linear Association

Theorem A.4 can be shown to apply to parallel machines because the expected makespan can be calculated by a series of convolutions and maximum operators. This applies, however, only in the context of static allocations. Furthermore, if we adjust the load in such a way that criticalities are balanced as per Theorem RN9.2, they will remain balanced after applying the common factor. However, Theorem RN9.3 involves a due date that is not subject to the common factor, and thus the criticalities obtained initially are not likely to remain intact after taking the association into account. Another way to look at this issue is to observe that the due date may be critical too—indeed the due date *should* be critical with a probability of $SL = 1 - 1/m\gamma$. But if we were to subject the original due date to the common adjustment, it would become a random variable. So it would have to be replaced by another, fixed, value. But there is no guarantee that the criticalities of the machines would remain intact relative to the revised due date. Further research is needed to test whether optimizing for the unadjusted values and then revising the due date is a good heuristic.

Finally, by Theorem 6.7, the special dispatching case with exponential processing times would yield a stochastically minimal makespan even though the adjusted values are not exponential. A similar result applies in any case where stochastic dominance is obtained.

References

van den Akker, M., J.A. Hoogeveen, and S.L. Van De Velde (1999) "Parallel Machine Scheduling by Column Generation," *Operations Research* 47, 862-872.

Baker, K.R. and Merten, A.G. (1973) "Scheduling with Parallel Processors and Linear Delay Costs," *Naval Research Logistics Quarterly* 20, 793-804.

Bruno, J., P. Downey, and G. Frederickson (1981) "Sequencing Tasks with Exponential Service Times to Minimize the Expected Flow Time or Makespan," *Journal of the Association for Computing Machinery* 28, 100-113.

Chang, S.Y. and H.-C. Hwang (1999) "The Worst-Case Analysis of the MULTIFIT Algorithm for Scheduling Nonsimultaneous Parallel Machines," *Discrete Applied Mathematics* 92, 135-147.

Chen, Z.-L. and W.B. Powell (1999) "Solving Parallel Machine Scheduling Problems by Column Generation," *INFORMS Journal of Computing* 11, 78-94.

Coffman, E.G., M.R. Garey, and D.S. Johnson (1978) "An Application of Bin Packing to Multiprocessor Scheduling," *SIAM Journal of Computing* 7, 1-17.

Coffman, E.G. and R.L. Graham (1972) "Optimal Scheduling for Two Processor Systems," *Acta Informatica* 1, 200-213.

Conway, R.W., W.L. Maxwell and L.W. Miller (1967) *Theory of Scheduling*, Addison-Wesley, Reading, Mass.

Davis, E. and J.M. Jaffe (1981) "Algorithms for Scheduling Tasks on Unrelated Processors," *Journal of the Association for Computing Machinery* 28, 721-736.

Dobson, G. (1984) "Scheduling Independent Tasks on Uniform Processors," *SIAM Journal of Computing* 13, 705-716.

Eastman, W.L., S. Even, and I.M. Isaacs (1964) "Bounds for the Optimal Scheduling of *n* Jobs on *m* Processors," *Management Science* 11, 268-279.

Friesen, D.K. and M.A. Langston (1986) "Evaluation of a Multifit-based Scheduling Algorithm," *Journal of Algorithms* 7, 35-59.

Ghirardi, M. and C.N. Potts (2005) "Makespan minimization for scheduling unrelated parallel machines: A recovering beam search approach," *European Journal of Operational Research* 165, 457-467.

Graham, R.L. (1969) "Bounds on Multiprocessor Timing Anomalies," *SIAM Journal of Applied Mathematics* 17, 416-425.

He, Y. (1999), "A Multifit Algorithm for Set Partitioning Containing Kernels," *Applied Mathematics Journal of Chinese Universities* 14B, 227-232.

Horn, W.A. (1973) "Minimizing Average Flow-Time with Parallel Machines," *Operations Research* 21, 846-847.

Horowitz, E. and S. Sahni (1976) "Exact and approximate algorithms for scheduling nonidentical processors," *Journal of the Association for Computing Machinery* 23, 317–327.

Hu, T.C. (1961) "Parallel Sequencing and Assembly Line Problems," *Operations Research* 9, 841-848.

Ibarra, O.H and C.E. Kim (1977) "Heuristic Algorithms for Scheduling Independent Tasks on Nonidentical Processors," *Journal of the Association for Computing Machinery* 24, 280-289.

Kao, T.Y. and E.A. Elsayed (1990) "Performance of the LPT Algorithm in Multiprocessor Scheduling," *Computers and Operations Research* 17, 365-373.

Lam, S. and R. Sethi (1977) "Worst Case Analysis of Two Scheduling Algorithms," *SIAM Journal of Computing* 6, 518-536.

Lee, C.-Y. (1991) "Parallel Machines Scheduling with Nonsimultaneous Machine Available Time," *Discrete Applied Mathematics* 30, 53-61.

Lee, C.-Y, Y. He and G. Tang (2000) "A Note on 'Parallel Machines Scheduling with Nonsimultaneous Machine Available Time'," *Discrete Applied Mathematics* 100, 133-135.

Lee, C.-Y. and J.D. Massey (1988) "Multiprocessor Scheduling: Combining LPT and Multifit," *Discrete Applied Mathematics* 20, 233-242.

Martello, S., F. Soumis and P. Toth (1997) "Exact and Approximation Algorithms for Makespan Minimization on Unrelated Parallel Machines," *Discrete Applied Mathematics* 75, 169–188.

McNaughton, R. (1959) "Scheduling with Deadlines and Loss Functions," *Management Science* 6, 1-12.

Mokotoff, E. and P. Chrétienne (2002) "A Cutting Plane Algorithm for the Unrelated Parallel Machine Scheduling Problem," *European Journal of Operational Research* 141, 515-525.

Pinedo, M. (2002) *Scheduling: Theory, Algorithms, and Systems*, 2nd edition, Prentice-Hall.

Portougal, V. (1993), "Asymptotic Behavior of some Scheduling Algorithms," *Asia-Pacific Journal of Operational Research* 10, 71-91.

Potts, C.N. (1985), "Analysis of a Linear Programming Heuristic for Scheduling Unrelated Parallel Machines," *Discrete Applied Mathematics* 10 (1985) 155–164.

Van de Velde, S.L. (1993) "Duality Based Algorithm for Scheduling Unrelated Parallel Machines," *ORSA Journal on Computing* 5, 192–205.

Weber, R.R. (1982) "Scheduling Jobs with Stochastic Processing Requirements on Parallel Machines to Minimize Makespan or Flowtime," *Journal of Applied Probability* 19, 167-182.

Weber, R.R., P. Varaiya, and J. Walrand (1986) "Scheduling Jobs with Stochastically Ordered Processing Times on Parallel Machines to Minimize Expected Flowtime," *Journal of Applied Probability* 23, 841-847.

Weiss, G. (1992) "Turnpike Optimality of Smith's Rule in Parallel Machines Stochastic Scheduling," *Mathematics of Operations Research* 17, 255-270.