Computational Results for the Flowshop Tardiness Problem

Kenneth R. Baker

Abstract

This paper reports on computational experiments involving optimal solutions to the flowshop tardiness problem. Of primary interest was a generic approach: solutions were obtained using a spreadsheet-based, mixed-integer programming code. However, the results compare favorably with those from a specially-tailored branch and bound algorithm. The main implication is that hardware and software have developed to the point that generic tools may offer the best way to solve combinatorial problems in scheduling.

Keywords: Scheduling, sequencing, flowshop, tardiness, integer programming, spreadsheet models, Risk Solver Platform

Computational Results for the Flowshop Tardiness Problem

In this paper we address the minimization of total tardiness in the permutation flowshop. This problem has been the subject of research in the past, but progress on finding optimal solutions has been limited. The computational results summarized here are arguably the best yet reported. However, the more important point is that they have been achieved without a tailored solution algorithm. Instead, the optimization problem was formulated as a mixed-integer program and solved with spreadsheet-based optimization software. The implication is that hardware and software have advanced to the point that generic tools are becoming competitive when it comes to solving some difficult scheduling problems.

This trend has important consequences for scheduling practitioners. The opportunity to use generic tools to solve complex scheduling problems is a major advantage. For practical scheduling problems, it may not be necessary to search the literature for a highly specialized, state-of-the-art solution algorithm when a generic solution may work just fine. A generic solution approach (that is, a spreadsheet-based formulation together with a publicly available mixed-integer programming code) is usually more accessible than a highly specialized algorithm. Moreover, it may not be easy to determine what problem sizes are amenable to solution by state-of-the-art algorithms if published results are out of date. (They are typically not updated as hardware tools develop.) For a generic approach, however, it is not difficult to keep a sample integer programming formulation on hand and test its solution with each new generation of hardware or software. Thus, the main implication is that spreadsheet-based optimization is becoming a viable solution tool for scheduling applications.

This result also has implications for scheduling researchers. When specialized algorithms are developed for combinatorial scheduling problems, it makes sense to use mixed-integer programming as a benchmark solution procedure, especially for cases in which algorithm development has been limited. In an earlier era, scheduling problems were sometimes solved using integer programming in order to gain insight into integer programming methods. Now, finally, the opposite seems to be true: integer programming methods can reveal our ability to solve the scheduling problems themselves.

Minimizing total tardiness in the flowshop model

The flowshop model contains n jobs and $m \geq 2$ machines (referred to as an nxm problem). Job j has a given processing time, p_{ij}, on machine i and a given due date, d_j. Each job must visit the machines in the same machine order, and the machines can process at most one job at a time. As a result of scheduling decisions, job j achieves a completion time, C_j. Its tardiness is defined as $T_j = \max\{0, C_j - d_j\}$. The total tardiness in the schedule is $\sum T_j$, which is the objective to be minimized.

We consider only schedules in which the jobs are processed in the same order on all machines, a so-called *permutation schedule*. Although it is theoretically possible that a non-permutation schedule could be optimal, the search for an optimum is far more complicated in the general case, and permutation schedules are much easier to implement in practice. For this reason, attention has been focused on permutation schedules. In standard notation, the problem we address is denoted $F|prmu|\sum T_j$.

This problem is *NP*-hard, as shown by Koulamas (1994). It can be viewed as a generalization of the single-machine tardiness problem (which is also *NP*-hard) or a generalization of the flowshop total completion time problem (which is again *NP*-hard). Because the flowshop tardiness problem is a generalization of other *NP*-hard problems, it comes as no surprise that finding solutions to problem instances of even modest size may be quite challenging. A few papers have described optimization algorithms for this problem, but most of the work on it has been oriented to heuristic solutions that do not guarantee optimality. In this paper, we focus on optimal solutions to the problem.

Progress in finding optimal solutions

Even the two-machine version of the flowshop tardiness problem is *NP*-hard, and for that reason, some research studies have addressed only that case. Other studies have investigated versions with more than two machines, where we would expect that solutions would be computationally even more difficult to obtain. The article by Vallada et al. (2008), which primarily covers heuristic methods, provides a careful review of previous research on the flowshop tardiness problem. Part of this literature is specialized to the two-machine model, but we focus here on the general case characterized by more than two machines. The state of the art has been represented in two papers.

Kim (1995) examined problems with different values of *m* and was able to solve problems as large as 14x4 and 13x8 within a time limit of one hour. Roughly a decade later, Chung et al. (2006) also considered different values of *m* and were able to solve most problems containing 15 jobs (and as many as 8 machines), but several 20-job instances in their testbed went unsolved. Instead of using a time limit, they terminated their algorithm on the basis of nodes visited in the branch-and-bound (BB) algorithm, using a limit of four million nodes.

Based on the results of these studies, the state of the art among specialized BB algorithms appears to be the solution of problems with up to about 15 jobs and 8 machines. Clearly, improvements in hardware have occurred since the Chung paper appeared, but that does not necessarily mean that we should expect to solve much larger problems today. Besides, estimating the largest problem size that could be solved by a specialized algorithm is somewhat complicated by the choice of a programming environment and the design of a testbed, as we discuss later.

Using these results as guideposts, we implemented a spreadsheet-based optimizing approach for comparison with the BB algorithm described by Chung et al. We formulated the flowshop tardiness problem as a mixed-integer program (see Appendix A for formulation details) and found solutions using Risk Solver Platform (RSP). RSP is an Excel add-in developed by Frontline

Systems, Inc. It is available with several textbooks and is widely used by students and practitioners.[1] We formulated the mixed integer program on an Excel spreadsheet, using Visual Basic for Applications (VBA) to construct the detailed model, and then invoked RSP from a VBA subroutine.

We implemented the BB algorithm (see Appendix B) in VBA as well, using Excel to provide input data and summarize the results. We used VBA's MicroTimer function to track solution times. Thus, our comparisons relied on a spreadsheet platform and the VBA programming language. For hardware, we used an Intel Core i7 2.7GHz processor with 8GB of RAM—that is, a laptop that would be typical of what many undergraduate students use these days. Finally, we designed a set of experiments to evaluate and compare the spreadsheet-based approach with the tailored BB algorithm.

Parameters of the test data

Most computational studies of tardiness problems have used the *tardiness factor* and the *due-date range* as parameters to guide the generation of random test problems. This pair of parameters was first used in combination by Baker and Martin (1974) for the single-machine problem and has been used in various studies of tardiness problems ever since. In the flowshop tardiness problem, these same two factors were used in the studies cited above, although sometimes with different interpretations. We review the details briefly.

The tardiness factor (TF) represents the expected fraction of tardy jobs in a randomly-chosen sequence. In the single-machine model, the expected makespan is $M = np$, where p denotes the mean processing time. Suppose we set the mean due date $d = kp$ (with $k \le n$). Then we expect k jobs to be on time in a random schedule, and we expect the fraction of tardy jobs to be $\text{TF} = 1 - k/n$. Thus, the average due date can be expressed as follows:

$$d = kp = n(1 - \text{TF})p = M(1 - \text{TF})$$

We can also represent the due-date range (DDR) as a fraction of the expected makespan. Assuming a symmetric distribution, the minimum due date is $d - M(\text{DDR})/2$ and the maximum is $d + M(\text{DDR})/2$. For sampling purposes, we can draw due dates from a uniform distribution between these two limits. In this derivation, the interval is based on the mean value p.

In the flowshop problem, no formula exists for the expected makespan. Reasoning with mean values, we can approximate the expected makespan as the total processing time on the first machine plus the total processing time for the last job after it finishes on the first machine. In other words,

$$M = np + (m - 1)p = (n + m - 1)p \qquad (1)$$

[1] In particular, the experiments reported here used default settings in the software, which includes implementation of the Gurobi Solver Engine.

The due dates can then be sampled from the interval [$d - M$(DDR)/2 , $d + M$(DDR)/2]. An equivalent expression for this interval is the following:

$$\text{Minimum due date} = M(1 - \text{TF} - \text{DDR}/2) \quad (2a)$$

$$\text{Maximum due date} = M(1 - \text{TF} + \text{DDR}/2) \quad (2b)$$

This method was used in our experiments. We first drew mn processing times from a uniform distribution on the integers from 1 to 99. Then, according to specified values of TF and DDR, we calculated the limits of the due-date distribution from (2a) and (2b) and drew n due dates from a discrete uniform distribution with those limits. After sampling, we computed optimal solutions using BB and RSP.

For the m-machine flowshop model, Kim (1995) used essentially the same method for generating test problems, except for using a lower bound on the on the estimated makespan described in (1). On the other hand, Chung et al. (2006) interpreted the parameters quite differently. In their experiments, the processing times were generated in several different ways, some with correlation, some with trend, some with both, and some with neither. If we consider just the case of neither (i.e., independent sampling of processing times), they essentially set $M = \gamma nmp$, where γ = 0.5, 1.0, or 1.5, and the other parameters were as defined above. In other words, aside from the factor γ, they interpreted M as the sum of *all* processing times. Obviously, if we think of this value as an estimate of the makespan, it is biased upward. (In the single-machine problem, of course, the sum of all processing times and the makespan are identical, but that equivalence does not hold in the flowshop problem.) Consider, for example, the case of n = 15 jobs and m = 4 machines, with TF = 0.65 and DDR = 0.4, corresponding to an intermediate sampling case in the data generated by Chung et al. Our formulas (1) and (2ab) would lead to M = 900 and an interval for sampling due dates of [135, 495]. In the dataset generated by Chung et al., the three values of γ would yield the following sampling intervals:

[225, 825] [450, 1650] [675, 2475]

In other words, those sampling intervals tend to be somewhat larger and shifted upward as compared to the intervals used in most other studies. Even in the first case (225 to 825), the interval corresponds to TF = 0.42 as most authors would define it. (See Vallada et al. 2008, for example.) As a result, their reported values of TF and DDR are not directly comparable to those in the other papers. In addition, the large value for the upper limit of their due-date distribution suggests that many of the instances in their study contained jobs with due dates larger than the makespan. (Nearly half of the parametric cases exhibit this property.) The corresponding jobs can be placed last in sequence, each time reducing the effective problem size, so it remains unclear whether the test problems adequately describe the problem sizes that the algorithm will handle. The comparisons, however, still demonstrate that the algorithm of Chung et al. performs better than its predecessor.

Computational results

We followed the design of (TF, DDR) pairs used for example by Potts & Van Wasssenhove (1985) and Szwarc et al. (2001) by taking DDR = {0.2, 0.4, 0.6, 0.8, 1.0} and TF = {0.2, 0.4, 0.6, 0.8}. For a given problem size and for each possible combination of DDR and TF, we drew four random instances. We then found optimal solutions to each of those 80 instances using RSP and BB. We repeated that design for several problem sizes from 10x4 to 16x8. Our summary statistics are the average, median, and maximum cpu time required to solve the test problems of each given size. The results are provided in Table 1.

Problem Size	***Average Time (Seconds)***		***Median Time (Seconds)***		***Maximum Time (Seconds)***	
	BB	RSP	BB	RSP	BB	RSP
10x4	0.67	0.46	0.51	0.40	3.72	1.41
10x8	2.56	1.24	2.39	1.19	10.72	2.73
12x4	7.72	0.96	3.23	0.78	88.97	4.21
12x8	23.04	4.54	17.16	3.95	115.86	22.86
14x4	91.51	3.39	28.65	2.24	979.82	48.86
14x8	235.55	18.80	143.90	13.53	1223.57	94.24

Table 1. Summary of computational results.

In the tabulated data, the most meaningful summary statistics may be the median times. When instances are relatively small and amenable to solution in seconds, the median time and the average time are relatively close together. As problem sizes get larger, a recognizable combinatorial pattern sets in. The average begins to grow faster than the median, and the maximum time grows far more quickly. We can only speculate about whether the observed maxima faithfully indicate how large the solution times can become. In their flowshop experiments, Companys and Mateo (2007) comment that sample sizes of even 1000 may not produce representative values for the maximum time. However, it is clear from the table that this "knee of the curve" phenomenon sets in earlier for BB than for RSP.

In the table, for every pairwise comparison between BB and RSP statistics, the RSP time is faster. Moreover, the RSP time appears to be more insulated from the combinatorial increase of solution time with problem size, at least in this range of problem sizes. In most cases, the average, median, and maximum solution times for the BB algorithm are an order of magnitude longer than those for the RSP approach. Ultimately, this means that, for a given computing capability, the largest problem we can handle will be larger for RSP than it will be for BB.

For problem sizes with more than 15 jobs, we can expect that some solution times might grow quite long. (Indeed, this was also the finding in Chung et al., even with the biases present in their sampling scheme.) A useful metric in this situation is the largest problem size for which a solution can be

obtained with less than an hour of cpu time. Table 2 shows additional results that shed light on this aspect of the comparison. Problem sizes of 16 jobs are too large for the BB algorithm to reliably find solutions within an hour of computer time, whereas RSP handles all of the 16-job problems, all but one of the 8-machine problems with 18 jobs, and all but two of the 4-machine problems with 22 jobs. Thus, RSP insulates computation time from the effects of problem size more than the BB algorithm.

Problem Size	***Average Time (Seconds)***		***Median Time (Seconds)***		***Percent Unsolved***	
	BB	RSP	BB	RSP	BB	RSP
16x4	809.21	10.80	214.38	5.90	15.0%	0
16x8	1950.93	163.70	1740.12	68.40	27.5%	0
18x4		27.96		13.07		0
18x8		860.81		548.43		2.5%
20x4		184.40		32.50		0
22x4		843.47		121.44		5.0%

Table 2. Results for larger problems.

Summary and Conclusions

This paper examined computational aspects of solving the flowshop tardiness problem. We compared the best specially-tailored algorithm in the literature with a generic, mixed-integer programming approach, using a spreadsheet-based platform in both cases. Our main observation is that the spreadsheet-based approach is at least competitive with (and usually better than) a specialized branch-and-bound approach. For the practitioner who faces a combinatorial scheduling problem, it may not be the best idea to scour the literature for an appropriately tailored solution algorithm and then to implement that algorithm in code. Instead, a generic approach that relies on spreadsheets and general-purpose optimization software might be the better computational choice. Advances in hardware and software have made spreadsheet-based optimization into a solution tool that competes with tailored, state-of-the-art algorithms. This phenomenon expands the population of potential problem solvers and thereby expands the set of problems that can be optimized in practice.

For practitioners, the implication is that mixed-integer programming is becoming an important tool for solving scheduling problems. For researchers, the implication is that mixed-integer programming should be considered a viable solution method. As new algorithms are invented and tested, mixed-integer programming should be treated as a relevant benchmark in justifying the efficiency and effectiveness of any newly-proposed, specialized algorithm.

References

Baker, K. and J. Martin (1974). An experimental comparison of solution algorithms for the single-machine tardiness problem. *Naval Research Logistics Quarterly* 21, 187-199.

Chung, C., J. Flynn and O. Kirca (2006). A branch and bound algorithm to minimize to total tardiness for *m*-machine permutation flowshop problems. *European Journal of Operational Research* 174, 1-10.

Companys, R. and M. Mateo (2007). Different behaviour of a double branch-and-bound algorithm on Fm|prmu|C_{max} and Fm|block|C_{max} problems. *Computers & Operations Research* 34, 938–953.

Kim, Y. (1995). Minimizing total tardiness in permutation flowshops. *European Journal of Operational Research* 85, 541-555.

Koulamas, C. (1994). The total tardiness problem: review and extensions. *Operations Research* 42, 1025-1041.

Potts, C.N. and L.N. Van Wassenhove (1985). A branch and bound algorithm for the total weighted tardiness problem. *Operations Research* 33, 363-377.

Stafford, E., F. Tseng and J. Gupta (2005). Comparative evaluation of MILP flowshop models. *Journal of the Operational Research Society* 56, 88-101.

Szwarc, W., A. Grasso and F. Della Croce (2001). Algorithmic paradoxes of the single-machine total tardiness problem. *Journal of Scheduling* 4, 93-104.

Vallada, E., R. Ruiz and G. Minella (2008). Minimising total tardiness in the *m*-machine flowshop problem: A review and evaluation of heuristics and metaheuristics. *Computers & Operations Research* 35, 1350-1373.

Appendix A. The Mixed-Integer Programming Approach

Two distinct formulation approaches to flowshop problems have appeared in the literature. In one, the key decision variable is $x_{jk} = 1$ if job j appears in sequence position k. This approach builds on a portion of the model that contains an assignment problem, so we refer to it as the *assignment formulation*. In the second approach, the key decision variable is $x_{jk} = 1$ if job j precedes job k in the sequence. We refer to it as the *precedence formulation*.

The two formulations have been compared for the makespan objective. The most persuasive study is due to Stafford et al. (2005), who reported computational comparisons demonstrating that the assignment formulation requires less computation time than the precedence formulation. They also examined variations within both types of formulations but did not find enough evidence to favor one variation over another.

For the tardiness objective, no such comparison has been made, but we expect that the central conclusions reached by Stafford et al. should still hold. In other words, the assignment formulation should be better than the precedence formulation. (Similar conclusions have been reached in studies of single-machine sequencing models.) We did, however, compare two variations. The first variation, which is the basis for our reported computation times, is called *the idle-time version* of the assignment formulation. The second variation is called the *completion-time version*.

A1. Idle-Time Version

In the idle-time version of the assignment formulation, the key supplementary variables and constraints involve idle times for jobs and machines.

Notation

n = number of jobs
m = number of machines
p_{ij} = processing time for job j on machine i
d_j = due date for job j

Decision variables

$x_{jk} = 1$ if job j is assigned to sequence position k, and zero otherwise (binary variable)
t_k = tardiness of the job in sequence position k
I_{ik} = idle time on machine i prior to the start of the job in sequence position k
H_{ki} = idle time for the job in sequence position k after finishing on machine i

Constraints

The assignment constraints require $\sum_{j=1}^{n} x_{ij} = 1$ and $\sum_{i=1}^{n} x_{ij} = 1$. Then, for all sequence positions k, $1 \le k \le n-1$, and all machines $1 \le i \le m-1$, we have the definitional relation between the idle time variables:

$$I_{i,k+1} + \sum_{u=1}^{n} p_{iu} x_{u,k+1} + H_{i,k+1} - H_{i,k} - \sum_{u=1}^{n} p_{i+1,u} x_{u,k} - I_{i+1,k+1} = 0$$

For all machines $1 \le i \le m-1$, we also have the relation for the first sequence position:

$$I_{i1} + \sum_{u=1}^{n} p_{iu} x_{u1} + H_{i1} - I_{i+1,1} = 0$$

Next, the completion time, C_k, of the job in position k, can be expressed as follows:

$$C_k = C_{k-1} + I_{mk} + \sum_{j=1}^{n} p_{mj} x_{jk}$$

where we can take C_0=0. The due date, d_k, of the job in position k, can be expressed as

$$d_k = \sum_{j=1}^{n} d_j x_{jk}.$$

Therefore, the tardiness of the job in position k must satisfy $t_k \geq C_k - d_k$, or

$$t_k \geq C_{k-1} + I_{mk} + \sum_{j=1}^{n} p_{mj} x_{jk} - \sum_{j=1}^{n} d_j x_{jk}$$

Objective: $Minimize \sum_{k=1}^{n} t_k$.

Model size: $n(n+2m+1)$ variables, of which n^2 are binary, and $3n+(m-1)n$ constraints.

A2. Completion-Time Version

In the *completion time version* of the assignment formulation, the key supplementary variables involve job completion times.

Notation

n = number of jobs
m = number of machines
p_{ij} = processing time for job j on machine i
d_j = due date for job j

Decision variables

x_{jk} = 1 if job j is assigned to sequence position k
t_k = tardiness of the job in sequence position k
C_{ik} = completion time on machine i for the job in sequence position k (with C_{0i} = 0 and C_{i0} = 0)

Constraints

Again, the assignment constraints require $\sum_{j=1}^{n} x_{ij} = 1$ and $\sum_{i=1}^{n} x_{ij} = 1$, and the due date, d_k, of the job in position k, can be expressed as $d_k = \sum_{j=1}^{n} d_j x_{jk}$. The completion times must satisfy a pair of constraints:

$$C_{ik} \geq C_{i-1,k} + \sum_{j=1}^{n} p_{ij} x_{jk}$$
$$C_{ik} \geq C_{i,k-1} + \sum_{j=1}^{n} p_{ij} x_{jk}$$

Finally, the tardiness of the job in position k must satisfy $t_k \geq C_k - \sum_{j=1}^{n} d_j x_{jk}$.

Objective: $Minimize \sum_{k=1}^{n} t_k$.

Model size: $n(n+m+1)$ variables, of which n^2 are binary, and $3n+m(2n-1)$ constraints.

In a set of preliminary tests, we implemented the experimental design for both variations, the idle-time variation and the completion-time variation. Consistent with the observations of Stafford et al., we did not observe that one formulation dominated. In some samples, one variation was better and in other samples, the other variation was better. However, most of the time, the idle-time variation generated superior performance, so we have reflected those results in our summaries. If we had reported times for the faster of the two variations, then clearly our summary data would be somewhat better, but we expect that practitioners and researchers implementing a mixed-integer programming approach are likely to select one formulation strategy and stick with it. That was the approach behind our reported experimental results.

Appendix B. The Branch and Bound Approach

The most effective branch and bound (BB) algorithm for the flowshop tardiness problem is due to Chung et al. (2006). Our experiments implemented their algorithm, which is described here.

B1. Branching structure. The BB algorithm is structured around a branching tree that could, in principle, generate all partial and full sequences for the problem. The nodes of the tree each correspond to a partial sequence of s jobs, where $1 \le s \le n$. The partial sequences are built from the start of the sequence, in a "forward" direction, so that the ith job in the partial sequence occupies the ith position on each machine. (The initial node corresponds to a null sequence and contains no jobs.) At level 1, the nodes correspond to one-job sequences; at level 2, the nodes correspond to two-job sequences, and so on. The generation of partial sequences follows the usual backtracking method, generating partial sequences in the order 1, 1-2, 1-2-3, and so on. Furthermore, the algorithm possesses the ability to prune dominated nodes by identifying pairs of partial sequences containing the same jobs and using the dominance conditions derived by Chung et al.

B2. Lower bound calculation. Nodes are also pruned if the corresponding lower bound is larger than the value of tardiness for a previously-encountered complete sequence. The lower bound is a machine-based calculation. Each machine in turn is assumed to be the only bottleneck operation. For machine j, it is possible to compute a lower bound on the start time of the ith job on machine j, using the recursive calculation given by Chung et al. From this bound, it is possible to compute a lower bound on the tardiness that could be achieved by completing the partial sequence.

B3. Implementation. Chung et al. also discuss the advantages of their branching and bounding structure, in comparison to other approaches in the literature. Their implementation produces what they call an adaptive, depth-first plus backtracking search strategy employing a dominance test and a lower bound. Prior to the BB phase, their algorithm produces a complete solution using a heuristic method. The algorithm reported in this paper follows the same design principles.