Computers & Industrial Engineering 64 (2013) 812–816

Contents lists available at SciVerse ScienceDirect

Computers & Industrial Engineering

journal homepage: www.elsevier.com/locate/caie

Computational results for the flowshop tardiness problem

Kenneth R. Baker

Tuck School of Business, Dartmouth College, Hanover, NH, United States

ARTICLE INFO

Article history:
Received 10 July 2012
Received in revised form 31 October 2012
Accepted 19 December 2012
Available online 7 January 2013

Keywords:
Scheduling
Sequencing
Flowshop
Tardiness
Integer programming
Spreadsheet models

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.

1. Introduction

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.

2. Minimizing total tardiness in the flowshop model

The flowshop model contains n jobs and $m \geqslant 2$ machines (referred to as an $n \times m$ 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

E-mail address: ken.baker@dartmouth.edu

0360-8352/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.cie.2012.12.018

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.

3. 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, Ruiz, and Minella (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 14×4 and 13×8 within a time limit of 1 h. Roughly a decade later, Chung, Flynn, and Kirca (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 2.7 GHz processor with 8 GB 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.

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

4. 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 $k \leqslant n$). Then we expect k jobs to be on time in a random schedule, and we expect the fraction of tardy jobs to be $\mathrm{TF} = 1 - k/n$. Thus, the average due date can be expressed as follows:

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

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 $d - M(\mathrm{DDR})/2$ and the maximum is $d + M(\mathrm{DDR})/2$. 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,

$$M = np + (m - 1)p = (n + m - 1)p \tag{1}$$

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

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

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

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) to (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 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 $\gamma = 0.5$, 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