Spreadsheet-based computations for the flowshop problem with synchronous transfers

Kenneth R. Baker

Tuck School of Business,
Dartmouth College,
Hanover, NH 03755, USA
Email: ken.baker@dartmouth.edu

Abstract: Even when the research literature makes available to us the latest, most sophisticated solution algorithms, it may be possible to satisfy our needs with a generic solution approach. Faced with the need to find a solution to a challenging sequencing problem, we might be well served by relying on off-the-shelf optimisation software, implemented in Excel. To illustrate this point as it applies to sequencing problems, we revisit some results from the recent literature and compare the outcomes we can obtain with a specialised algorithm to those from a general-purpose, spreadsheet-based approach.

Keywords: sequencing; scheduling; flowshop; optimisation; heuristic; spreadsheet; integer programming.

Reference to this paper should be made as follows: Baker, K.R. (2014) 'Spreadsheet-based computations for the flowshop problem with synchronous transfers', *Int. J. Planning and Scheduling*, Vol. 2, No. 1, pp.77–86.

Biographical notes: Kenneth R. Baker is a faculty member at Dartmouth College. He is a currently Nathaniel Leverone Professor of Management at the Tuck School of Business and an Adjunct Professor of Engineering at the Thayer School of Engineering. His recent textbooks include *Principles of Sequencing and Scheduling*, co-authored with Dan Trietsch, *Management Science*, co-authored with Stephen Powell, and *Optimization Modeling with Spreadsheets* (Second Edition), all published by John Wiley & Sons. He is a member of the journal's editorial board.

1 Introduction

The flowshop model is a basic and widely-studied scheduling model. A special variation of the model, involving synchronous material movement, has recently been highlighted in the work of Huang and Ventura (2013), hereafter referred to as H&V. They created and tested an original dynamic programming algorithm that finds the minimum makespan in a three-station flowshop with synchronous transfers. Computational results indicated that their algorithm was ultimately limited by its storage requirements, but they were able to produce optimal solutions for problems containing up to 17 jobs. They also developed and tested several heuristic methods for the same problem, indicating that those methods could be effective for problem sizes that the dynamic programming algorithm could not accommodate.

In the inaugural issue of this journal, the case was made for attacking sequencing problems with a generic spreadsheet-based approach (Baker, 2011). In an earlier era, the spreadsheet was seldom considered a serious platform for solving sequencing problems. Personal computers and laptops were not particularly powerful, and the optimisation software that ran on those machines was not competitive with software available on larger machines. As a result, the spreadsheet was often dismissed as a simplistic mathematical device that might serve the educational environment but could not solve serious optimisation problems. In that setting, developments in scheduling theory favoured special-purpose solution methods that were tailored to particular types of problems. Such methods flourished in the research literature, and for good reason: they represented the best way – sometimes the only way – to solve challenging sequencing problems.

With the advance of technology, however, that situation has changed. Excel has become familiar to a wide audience, especially in the business world. More and more users have become comfortable using the Excel interface, typically for purposes other than optimisation. Hardware and software have improved considerably, and general-purpose optimisation methods have been tuned to perform well in spreadsheet environments. In particular, mixed-integer programming formulations of certain sequencing problems can be solved with generic spreadsheet-based optimisation software to produce results that in some cases virtually match the computational speed of specialised, state-of-the-art procedures.

A similar phenomenon has occurred in the development of heuristic solution methods. Off-the shelf versions of powerful, general-purpose heuristic algorithms have proven effective at solving sequencing problems on spreadsheets, and Excel users can employ these algorithms with modest preparation requirements.

The implication is that general-purpose software merits consideration when exploring ways of finding solutions to sequencing problems, either for specific practical cases or in research studies. For practitioners, it is not always necessary to solve the largest, most challenging version of a particular problem, and a spreadsheet-based approach may be the most sensible one. For researchers, the tradition of searching for ever more novel and clever tailored algorithms may overlook the impressive capabilities of spreadsheet-based software, thereby leaving a potential gap in the evaluation of computational approaches to solving a theoretical problem type. This paper reinforces that point by revisiting the flowshop problem with synchronous transfers and demonstrating the virtues of a generic solution approach.

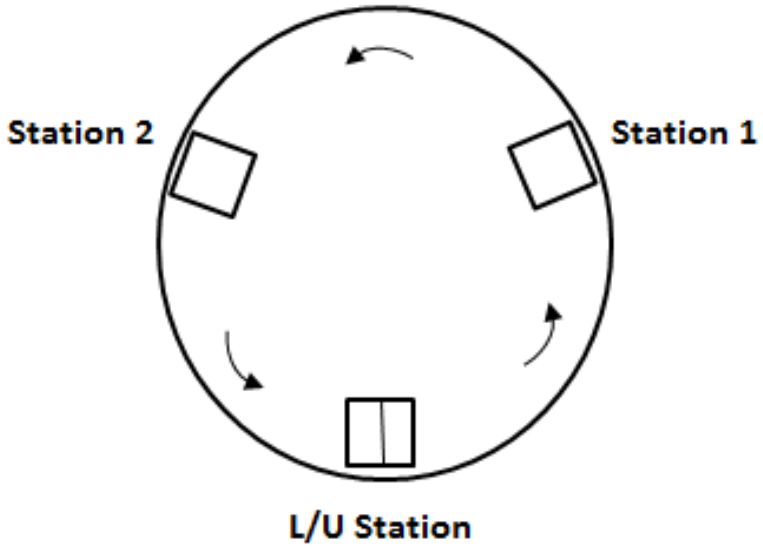
In the next section, we review the main features of the problem under study and describe a mixed-integer programming formulation. We then report some computational results using the kind of hardware and software that is easily accessible to students and practitioners. Those results show that the mixed-integer programme finds solutions roughly as fast as the dynamic programming algorithm, but it does not face storage limits. We then turn to the subject of heuristic solutions and present computational evidence of better solutions than were previously available.

2 The flowshop with synchronous transfers

In the H&V version of the problem, a machining centre contains three stations (see Figure 1). One station is a loading/unloading (L/U) station, where parts are placed into

the production line and later removed from the line. The two other stations each house a CNC machine. An individual part or job moves from the L/U station to the first CNC machine, then to the second CNC machine, and finally back to the L/U station, where it leaves the centre. We can think of the parts as travelling along a clock face from loading at 6 o'clock to a first operation at 2 o'clock to a second operation at 10 o'clock and then back to 6 o'clock to unload. The parts move around the machining centre on a rotary table that transfers all resident parts simultaneously.

Figure 1 Diagram of the synchronous flowshop



For the purposes of notation, let L_j be the loading time for job j and let U_j be the unloading time. In addition let A_j be the processing time for job j on the first CNC machine and let B_j be the processing time for job j on the second CNC machine. The schedule consists of a series of time intervals, or cycles, within which each job experiences one operation before the rotary table transfers it to the next station. If we let $[j]$ represent the job in sequence position j, then the cycle times y_j become

$$
\begin{aligned}
y_1 &= L_{[1]} \\
y_2 &= \max\left\{L_{[2]}, A_{[1]}\right\} \\
y_3 &= \max\left\{L_{[3]}, B_{[1]}, A_{[2]}\right\} \\
&\cdots \\
y_k &= \max\left\{L_{[k]} + U_{[k-3]}, B_{[k-2]}, A_{[k-1]}\right\}, \quad \text{for } 4 \le k \le n \\
&\cdots \\
y_{n+1} &= \max\left\{U_{[n-2]}, B_{[n-1]}, A_{[n]}\right\} \\
y_{n+2} &= \max\left\{U_{[n-1]}, B_{[n]}\right\} \\
y_{n+3} &= U_{[n]}
\end{aligned}
$$

The objective then becomes

$$M = \sum_{j=1}^{n+3} y_j \tag{1}$$

To formulate the problem of minimising the makespan as a mixed-integer programme, we use *assignment variables*, defined as $x_{ij} = 1$ if job i occupies sequence position j. Thus, a set of assignment constraints is necessary to ensure that a full sequence exists, or

$$\sum_{j=1}^{n} x_{ij} = 1, \quad i = 1, 2, \ldots, n \tag{2}$$

and

$$\sum_{i=1}^{n} x_{ij} = 1, \quad j = 1, 2, \ldots, n \tag{3}$$

In addition let y_k denote the k^{th} cycle time, as above, so that we require

$$y_k \geq \sum_{i=1}^{n} L_i x_{ik} + \sum_{i=1}^{n} U_i x_{i,k-3}, \; k = 1, 2, \ldots, n+3 \tag{4}$$

$$y_k \geq \sum_{i=1}^{n} A x_{i,k-1} \; k = 2, 3, \ldots, n+1 \tag{5}$$

$$y_k \geq \sum_{i=1}^{n} B x_{i,k-2}, \; k = 3, 4, \ldots, n+2 \tag{6}$$

Here we define $x_{ij} = 0$ for $j < 0$ or $j > n$. The problem can thus be posed as the mixed-integer programming problem of minimising (1) subject to (2) to (6), with the x_{ij} variables as binary variables. This formulation contains $(n^2 + n + 3)$ variables and $(5n + 3)$ constraints.

A spreadsheet layout for the ten-job problem is shown in Figure 2, with explanations provided in Figure 3. The model was built for solution using the Gurobi Engine option in Analytic Solver Platform (ASP). ASP is a well-known, spreadsheet-based software package used in many undergraduate and graduate courses as well as by practitioners. (More information can be found at http://www.solver.com).

The shaded cells in the display of Figure 2 are the decision variables, and these rows and columns would scale with the problem size. (The ten-job problem contains 113 variables, 100 of which are integers, along with 53 constraints). The specification of the ten-job optimisation problem is as follows:

Objective:	S3 (minimise)
Variables:	F3:R3
	F4:O13
Constraints:	F14:O14 = 1
	S4:S13 = 1
	F19 ≤ F3
	G19:G20 ≤ G3
	H19:H21 ≤ H3
	I19:I21 ≤ I3
	J19:J21 ≤ J3

K19:K21 ≤ K3
L19:L21 ≤ L3
M19:M21 ≤ M3
N19:N21 ≤ N3
O19:O21 ≤ O3
P19:P21 ≤ P3
Q19:Q21 ≤ Q3
R19 ≤ R3
F4:O13 binary

Figure 2 Spreadsheet layout of the optimisation model (see online version for colours)

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1																			
2			Cycle Number			1	2	3	4	5	6	7	8	9	10	11	12	13	Z
3	Job	L	P1	P2	U	2	1	8	8	9	15	8	12	4	10	2	3	1	83
4	1	7	15	3	2	0	0	0	0	1	0	0	0	0	0				1
5	2	6	6	9	3	0	0	1	0	0	0	0	0	0	0				1
6	3	5	4	10	2	0	0	0	0	0	0	0	1	0	0				1
7	4	5	1	3	1	0	0	0	0	0	0	0	0	0	1				1
8	5	2	1	3	1	1	0	0	0	0	0	0	0	0	0				1
9	6	1	8	5	2	0	1	0	0	0	0	0	0	0	0				1
10	7	6	12	4	3	0	0	0	0	0	0	1	0	0	0				1
11	8	2	8	1	1	0	0	0	0	0	0	0	0	1	0				1
12	9	7	1	14	2	0	0	0	1	0	0	0	0	0	0				1
13	10	7	3	12	2	0	0	0	0	0	1	0	0	0	0				1
14						1	1	1	1	1	1	1	1	1	1				
15					L	2	1	6	7	7	7	6	5	2	5				
16					P1	1	8	6	1	15	3	12	4	8	1				
17					P2	3	5	9	14	3	12	4	10	1	3				
18					U	1	2	3	2	2	2	3	2	1	1				
19						2	1	6	8	9	10	8	7	4	8	2	1	1	
20							1	8	6	1	15	3	12	4	8	1	0		
21								3	5	9	14	3	12	4	10	1	3		

Figure 3 Legend for the spreadsheet model in Figure 2

A4:A13	Job number (1 to n)
B4:E10	Times for loading (L), unloading (U), and two operations (P1, P2)
S3	Objective function
F3:R3	Cycle times for each of the (n+3) cycles
F4:O13	Binary "assignment" variables specifying the sequence
S4:S13	Row sums (=1) for the assignment variables
F14:O14	Column sums (=1) for the assignment variables
F2:O2	Cycle number (1 to n+3); also Position in sequence (1 to n)
F15:O15	Loading times for job in this sequence position SUMPRODUCT(F$4:F$13,$B$4:$B$13), copied to the right
F16:O16	Time for operation 1 in this sequence position SUMPRODUCT(F$4:F$13,$C$4:$C$13), copied to the right
F17:O17	Time for operation 2 in this sequence position SUMPRODUCT(F$4:F$13,$D$4:$D$13), copied to the right
F18:O18	Unloading times for job in this sequence position SUMPRODUCT(F$4:F$13,$E$4:$E$13), copied to the right
F19:O19	Cycle time required for loading and/or unloading in this cycle
F20:O20	Cycle time required for operation 1 in this cycle
F19:O21	Cycle time required for operation 2 in this cycle

For the purposes of evaluating the spreadsheet-based model, a set of test problems was created, following the H&V experimental design. Three different regimes were covered in which processing times were sampled from a discrete uniform distribution. The lower limit of these distributions was always 1; the upper limits are shown in Table 1. Scenario I represents a balanced case, in which the expected total of loading and unloading times matches the expected times on both CNC machines. In Scenario II, the processing times dominate, meaning that their expected values are greater than the expected total of loading and unloading times. In scenario III, the opposite is true: the processing times are dominated by the sum of loading and unloading times.

Table 1 Parameters for constructing problem instances

Scenario	*L*	*A*	*B*	*U*
I	7	11	11	3
II	7	15	15	3
III	10	11	11	4

Ten problem instances were created for each scenario in the H&V experiments. In the case of their dynamic programming algorithm, however, solution times are likely to be fairly similar for a given number of jobs because the computational effort required by the algorithm is driven mainly by problem size. For mixed-integer programming, however, that is not the case. As with many combinatorial problems, the computational effort depends on the data in a given instance as well as on the problem size and can exhibit considerable variability for a given value of n. This characteristic suggests that it may be useful to capture median run times as well as average run times, and that larger samples are desirable in order to portray the performance of the algorithm. (H&V did not report median run times, but we might guess that they were quite close to average run times.) Thus, our experiments called for 30 instances for each scenario, or three times the number used by H&V.

Table 2 Comparative results for optimisation

Problem size	*Dataset*	*ASP average*	*ASP median*	*H&V average*
10	I	0.73	0.72	0.24
10	II	0.73	0.52	0.20
10	III	0.58	0.48	0.19
15	I	16.8	16.7	
15	II	10.4	7.2	
15	III	9.6	3.1	
17	I	380.2	172.8	201.2
17	II	678.2	69.2	200.6
17	III	293.0	11.0	202.0
19	I	922.6	423.7	
19	II	1,543.0	922.7	
19	III	1,213.1	53.7	

The experimental results are summarised in Table 2, which includes problem sizes of 10, 15, 17, and 19. Problem sizes of 10 and 17 were used in the work of H&V. Their reported average cpu times are reproduced in Table 2. (Their results were based on the use of a 1.6 Ghz processor versus a 2.9 Ghz processor in the present case.) All reported times are in seconds.

The results in Table 2 show that problems containing ten jobs can be solved in less than a second, on average, using either method, but with the dynamic programming algorithm providing faster run times. Problems containing 17 jobs are more challenging, with both algorithms taking an average of a few minutes. Here, the mixed-integer approach often exhibits larger average run times but smaller median run times.

Also shown in Table 2 are the results for problems containing 19 jobs for the mixed-integer approach. Problems of this size were beyond the capability of the H&V algorithm. In these experiments, the mixed-integer algorithm was terminated after an hour of run time. As a result, in nine of the 90 test problems, the solution procedure did not prove optimality, although a closer look demonstrated that in some cases the optimal makespan had been found. (In the literature on integer programming, it is sometimes observed that codes take longer to prove optimality than to find an optimal solution in the first place. That characterisation appears to apply to this formulation, at least as it relates to problem sizes of 17 and above). Thus, although there is no formal limit to the problem size that can be attacked with mixed-integer formulation, run times may occasionally grow quite large depending on the parameters in a specific problem. On the other hand, we can observe that in dataset III the run times were often quite short, exhibiting a median of less than a minute even for 19-job problems.

3 Heuristic solutions

In addition to its mixed-integer optimisation capability, ASP offers a generic evolutionary algorithm that has proven effective at solving sequencing problems. Building a spreadsheet-based model for this purpose is more straightforward than for mixed-integer programming, as evidenced by the model displayed in Figure 4. In the spreadsheet, rows 1-5 contain the data for a given instance. Row 7 contains the only decision variables, which assign jobs to positions in sequence. Rows 8 to 11 rearrange the various processing times to follow the sequence specified in Row 7. The cycle times can then be calculated in Row 17, along with their sum, thus providing the makespan, captured in cell C17. The model specification is as follows.

Objective: C17 (minimise)
Variables: D7:M7
Constraints: D7:M7 = alldifferent

Figure 4 Spreadsheet layout of the heuristic model (see online version for colours)

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	Position			1	2	3	4	5	6	7	8	9	10			
2			L	7	6	5	5	2	1	6	2	7	7			
3			P1	15	6	4	1	1	8	12	8	1	3			
4			P2	3	9	10	3	3	5	4	1	14	12			
5			U	2	3	2	1	1	2	3	1	2	2			
6																
7	Sequence			5	6	2	9	1	10	7	3	8	4			
8			L	2	1	6	7	7	7	6	5	2	5			
9			P1	1	8	6	1	15	3	12	4	8	1			
10			P2	3	5	9	14	3	12	4	10	1	3			
11			U	1	2	3	2	2	2	3	2	1	1			
12																
13	Cycle			1	2	3	4	5	6	7	8	9	10	11	12	13
14			L/U	2	1	6	8	9	10	8	7	4	8	2	1	1
15			1		1	8	6	1	15	3	12	4	8	1		
16			2			3	5	9	14	3	12	4	10	1	3	
17	Times		83	2	1	8	8	9	15	8	12	4	10	2	3	1

Solutions were generated using the genetic algorithm option for the Evolutionary Engine in ASP, which is suited to non-smooth models such as this one. The implementation of that option is most effective with manual intervention after many, if not all, individual runs of the procedure, although the specific intervention may often be subjective. However, that mode of iterative solution does not lend itself to the kinds of batch runs used earlier in the optimisation experiments. To explore the possibilities using a simple batch format, the following procedure was adopted.

1 Initialise by sequencing the jobs in numerical order.

2 Run the evolutionary engine using its default parameters and a 10-second time limit.

3 Re-run the evolutionary engine to see whether its stochastic elements produce an improvement. If so, continue to re-run the algorithm.

4 When a run does not improve the solution, re-initialise the sequence by taking the jobs in reverse numerical order, increase the mutation rate, and run the evolutionary engine.

5 Repeat step 3.

6 Select the better of the solutions generated from the two initialisation procedures.

The evolutionary engine contains random components. As a consequence, no two runs from a given initial sequence can be guaranteed to produce the same final solution. Similarly, the automated procedure used here may or may not produce a better solution than a flexible, manual implementation. However, this standardised implementation likely underestimates the results attainable by manual implementation, which can involve many tailored adjustments of the search parameters as well as longer run times.

The experimental results are summarised in Table 3, based on the same problem instances created for the optimisation runs in Table 2. For the evolutionary solver of ASP, Table 3 shows the average percentage error in the heuristic solution as well as the number of optima produced (out of 30). The average percentage error reported for the H&V

heuristics are those for the variation CAGI-S2, which, in the H&V experiments, produced the smallest average error in all cases except for dataset I and $n = 10$ (for which the best average among the H&V heuristics was 1.05%).

Table 3 Comparative results for heuristic solutions

Problem size	*Dataset*	*ASP error*	*H&V error*	*ASP opt.*
10	I	0.00%	1.25%	30
10	II	0.00%	1.15%	30
10	III	0.00%	0.72%	30
15	I	0.89%		10
15	II	0.36%		19
15	III	0.43%		19
17	I	0.84%	2.18%	9
17	II	0.44%	0.97%	11
17	III	0.61%	0.70%	15
19	I	1.38%		1
19	II	0.67%		9
19	III	0.93%		11

Although the instances were not the same in both studies, it does appear that the evolutionary algorithm in ASP tends to provide better performance where comparisons are possible. That is, its average suboptimality was lower than that of the CAGI-S2 heuristic in the six datasets that used common parameters. Noteworthy is that fact that optimal solutions to all of the ten-job problems were found by the evolutionary heuristic. For larger problems, the evolutionary heuristic produced an average suboptimality that increased with problem size, but even on larger problems, the average error remained small. For the 270 test problems with $n \leq 17$, the worst case error observed for the evolutionary heuristic was 2.2%, and it is worth reiterating that this figure likely underestimates the algorithm's potential because of the 10-secnd time limit and the simplified nature of the experimental method.

4 Conclusions

Special-purpose solution algorithms for challenging sequencing problems have been found throughout the literature on scheduling for most of the time it has been an area of research interest. However, adapting those algorithms for specific practical purposes can itself be a challenging task. In this paper, we reinforce the point that it is sometimes possible to avoid the demands of learning, tuning, and coding a highly specialised procedure to find solutions. Instead, it is possible to obtain satisfactory results with an approach that is both accessible and generic – using spreadsheet-based software with general-purpose optimisation capabilities. We used the synchronous flowshop model as a case in point and demonstrated the effectiveness of using a mixed-integer programming approach (for optimisation) or a standard evolutionary approach (for heuristic solutions).

These generic alternatives cannot promise superior performance in every instance, but they deserve to be considered seriously as suitable algorithms.

The results of this type helps bridge the gap between research and practice. A spreadsheet-based approach that uses general-purpose optimisation capabilities represents a methodology that can work in both settings. For researchers, this methodology represents an important candidate for solving computationally demanding problems and provides a useful perspective on the computational performance of special-purpose algorithms. For practitioners, this methodology allows for faster and easier model development, as compared to the use of special-purpose algorithms, and even gives practitioners a basis to conduct their own research investigations. Although mixed integer programming and evolutionary heuristics are well-established methodologies, they deserve serious consideration by both researchers and practitioners.

References

Baker, K. (2011) 'Solving sequencing problems in spreadsheets', *International Journal of Planning and Scheduling*, Vol. 1, No. 1, pp.3–18.

Huang, K-L. and Ventura, J.A. (2013) 'Two-machine flow shop scheduling with synchronous material movement', *International Journal of Planning and Scheduling*, Vol. 1, No. 4, pp.301–315.