Job Selection and Sequencing with Identical Tardiness Penalties

In the classical sequencing problem a set of available jobs exists, and the problem is to schedule those jobs in the presence of a limited resource. In a broader context, when the resource is particularly scarce, the workload itself becomes an issue. In that context, the first task is to select a subset of the available jobs to determine the workload; then the task is to develop a schedule for the selected subset. Problems that include both considerations thus integrate decisions regarding aggregate use of capacity with decisions regarding detailed use. In the research literature, such problems are called *order acceptance and scheduling* (OAS) problems. The interaction between job selection and detailed scheduling arises in many practical settings, as discussed by Roundy et al. (1995), Charnsirisakskul et al. (2004), Yang and Geunes (2007), and several other authors.

This paper deals with a special case of the OAS problem. Our model contains n jobs (or orders), simultaneously available for processing by a single machine. Job j is described by a processing time (p_j), a due date (d_j), and a revenue value (v_j). As a result of scheduling decisions, job j achieves a completion time (C_j), which may or may not meet its due date. Its tardiness (T_j) is defined in the usual way as $T_j = \max\{0, C_j - d_j\}$. At the outset, a decision must be made whether to accept or reject each job. Rejected jobs do not enter the system and are ignored, although they represent foregone revenue. Jobs j that are accepted provide revenue equal to v_j and must be sequenced. If job j finishes late, then it incurs a penalty cost equal to T_j. The measure of performance is thus the total profit (TP) attained by the schedule, or

$$TP = \sum_{j \varepsilon A} (v_j - T_j)$$

where A denotes the set of accepted jobs. From the customer's point of view, the cost of the order is mitigated by a discount that depends on the length of time the corresponding job is late, if it fails to meet its due date. In particular, the same unit tardiness penalty applies to all jobs, whereas the revenue values may differ.

More general versions of the OAS problem have been examined in the literature. As summarized in the next section, various papers have addressed the OAS problem with such additional features as release dates, deadlines, and setup times. Those formulations virtually always assume job-dependent tardiness penalties, and the consensus in those papers is that finding an optimal solution to the OAS problem is extremely difficult. A rough conclusion from that work is that 10-job problems still represent the computational frontier for optimizing several versions of the model. Therefore, most of the papers deal primarily with heuristic solution procedures. Roughly speaking, an effective heuristic procedure for this type of problem appears to be one that can find solutions within about 10% of the optimum on average.

In this paper, we pause to examine the special case in which unit tardiness penalties are identical. We do so based on an analogy with the single-machine tardiness problem (i.e., without job selection decisions). The classical tardiness problem is a special case of the single-machine problem with

weighted tardiness objective, and specialized techniques allow for the solution of much larger tardiness problems than weighted tardiness problems. If we want to solve tardiness problems, it does not make sense to use techniques that were designed for weighted tardiness problems, and we can't expect to obtain the best computational results using those techniques. For the OAS problem, we have already discovered that the weighted version is extremely difficult to solve, but little evidence has been developed about the solution of the special case with identical weights. Here, we focus on that special case and demonstrate that we can obtain optimal solutions to problems containing 25 jobs using a generic approach. Moreover, our heuristic solutions are suboptimal by an average of less than 1%. In other words, if we wanted to solve a 25-job problem to optimality, or get within a few percent of the optimum, we would likely be unsuccessful using the methods prescribed for the OAS problem with weights. However, we would succeed with the approaches described in this paper.

Specifically, we provide the results of computational experiments that cover optimal and heuristic solutions of the OAS problem with identical unit tardiness penalties. The approach relies on spreadsheet-based models and exploits the use of Risk Solver Platform (RSP), which is a state-of-the-art software package readily available to researchers, practitioners, and students.[1] Although its educational version is widely used in courses on Operations Research and related subjects, RSP has also emerged as a competitive computational tool for solving complex scheduling problems (Baker, 2011). Thus, we reinforce the potential significance of using traditional methods to tackle complicated scheduling problems. By relying on RSP, we intrinsically ensure that the modeling and solution methods are familiar and accessible to a broad audience of researchers and practitioners.

Related Research

A comprehensive review of related work is provided by Slotnick (2011), who describes the motivation for these kinds of models and classifies the various methodological results in the research literature related to the term "order acceptance and scheduling." Deterministic single-machine problems constitute one major category of research covered in that review. Models in that category may contain such features as setup times, job preemption, release dates, deadlines, controllable processing times, and resource constraints, although none of those assumptions occurs in our model. In particular, our model is a special case of the OAS problem first studied by Slotnick and Morton (2007) and by several other authors, but with equal unit tardiness penalties. The model is also a generalization of the OAS problem with weighted completion time penalties studied earlier by Slotnick and Morton (1996): in that case, the sequencing of the jobs was not in question, so the focus was mainly on job selection.

Slotnick and Morton (2007) found optimal solutions to the more general problem containing job-dependent tardiness penalties (or weights). In their sample of 100 ten-job problems, they were able to compute optimal solutions in an average of 1.7 hours of cpu time. When it came to 20-job problems, they stated that their optimizing procedure "is not practical for problems of this size," and they limited consideration to heuristic methods. Chen et al. (2008) dealt with a version of the model that includes

[1] For more information, see www.solver.com.

earliness costs along with tardiness costs and setup times as well. Their paper states that "no exact algorithm is capable of solving the optimization problem with reasonable computation time," and only heuristic methods are considered. Oguz et al. (2010) considered a version of the model with release dates, deadlines and setup times. The smallest problems in their computational experiments were 190 ten-job problems, but they were unable to solve several of those instances within an hour of cpu time. More recently, Cesaret et al. (2012) revisited the same model and created a set of 250 ten-job instances. All of those instances were solved using an integer programming approach, with an average run time of nearly 4 minutes. However, more than half of the 15-job instances were not solved within an hour of cpu time.

Each of the papers noted above dealt with job-dependent tardiness costs (and possibly other generalizations), but only in the most recent of the studies was it possible to reliably solve ten-job problems in an hour of cpu time. The findings suggest that optimization procedures are not viable except for small problem sizes. Thus, the motivating question for the present research is how much we can improve on this performance when we focus on the case of identical unit tardiness costs.

An alternative term for describing these models is “job scheduling with rejection,” in which penalties apply to rejected jobs. The terminology is usually attributed to Bartal et al. (2000) and has appeared in several more recent papers. A recent summary of those papers was compiled by Zhang et al. (2009), covering such criteria as makespan, total completion time, and weighted number of tardy jobs, in addition to rejection penalties. We can position our model in this literature as a single-machine scheduling problem with job-rejection and tardiness penalties, symbolically represented in standard notation as $1|\text{reject}|(\Sigma v + \Sigma T)$. In other words, the objective is to minimize the sum of two opportunity costs: the cost of lost revenue and the cost of missed due dates. In job-rejection segment of the literature, emphasis has primarily been placed on demonstrating NP-hardness and on developing polynomial-time approximation schemes. The actual computation of optimal solutions has seldom been addressed.

The OAS problem with identical unit tardiness penalties has not previously been specifically examined in the literature—neither in the OAS literature nor in the job-rejection literature. The case in which all jobs must be accepted corresponds, of course, to the problem of minimizing total tardiness, which is known to be NP-hard (Du and Leung, 1990). The more general case, and the closest analog in the literature, involves job-dependent tardiness penalties, giving rise to the problem of OAS with weighted tardiness penalties introduced by Slotnick and Morton (2007), which is of course also NP-hard.

Thus, we contribute to the research literature the first close look at the OAS problem with identical unit tardiness penalties. We describe optimization and heuristic approaches to solving the problem, and we provide computational results to elaborate on their algorithmic effectiveness. First, we describe the optimization approach and summarize the corresponding computational results. Then we describe three heuristic approaches and compare them computationally. A final section summarizes our conclusions.

Optimization Approach

To find optimal solutions to the OAS problem with tardiness penalties, we use a mixed integer programming (MIP) formulation. In fact, several formulation approaches have been suggested for sequencing problems, as reviewed in the case of tardiness-related measures by Keha et al. (2009) and by Baker and Keller (2010). The evidence indicates that the most efficient formulation exploits "sequence-position" variables—that is, binary variables indicating a position in sequence for each of the jobs. (For the generalization to job-dependent tardiness penalties, the sequence-position approach is more problematic, and only less efficient formulation structures are available.) We adapt this idea to the OAS problem with identical unit tardiness penalties as follows.

Decision variables

y_j = 1 if job j is accepted; otherwise, y_j = 0.

x_{jk} = 1 if job j is accepted and assigned to position k in sequence; otherwise, x_{jk} = 0.

t_k = tardiness of the k^{th} job in sequence.

Objective function

Maximize $\quad TP = \sum_{j=1}^{n} v_j y_j - \sum_{k=1}^{n} t_k$ (1)

Constraints

$$\sum_{k=1}^{n} x_{jk} = y_j, \quad \text{for } j = 1, 2, \ \ldots, n \tag{2}$$

$$\sum_{j=1}^{n} x_{jk} \leq 1, \quad \text{for } k = 1, 2, \ \ldots, n \tag{3}$$

$$\sum_{j=1}^{n} p_j \sum_{i=1}^{k} x_{ji} - \sum_{j=1}^{n} d_j x_{jk} \leq t_k, \quad \text{for } k = 1, 2, \ldots, n \tag{4}$$

$$y_j \ binary, x_{jk} \ binary, and \ t_k \geq 0$$

In this formulation, the objective function (1) sums the values of the accepted jobs and subtracts the sum of the tardiness levels for the late jobs. Constraint (2) states that accepted jobs must each be assigned a single position in sequence. Constraint (3) states that no sequence position can be assigned to more than one job. Constraint (4) states that the tardiness for the job in the k^{th} position must be at least as large as the difference between the completion time for the job in the k^{th} position and the due date for that job. (Because the model aims at maximizing *TP*, the incentive is to make t_k as small as possible while remaining nonnegative.)

Variations of this model are worth considering. For one, the y_j variables need not be treated as binary; they can be defined as continuous variables $0 \leq y_j \leq 1$. This formulation reduces the number of integer variables but adds to the number of constraints. Another variation expresses the objective function in (1) in terms of the variables x_{jk} and t_k, with coefficients of zero for y_j. These variations are ostensibly equivalent but sometimes lead to different run times. We return to this point later.

For a problem containing n jobs, the number of variables is $n(n + 2)$, of which $n(n + 1)$ are binary. In addition, the number of constraints is $3n$. Thus, for example, a 20-job problem gives rise to an MIP formulation containing 440 variables (420 of which are binary) and 60 constraints. As n grows, the formulation becomes larger and eventually requires a prohibitive computational effort. Our computational experiments suggest where that point might lie.

Test Problems

For the purposes of computational testing, we generated random instances of the OAS problem with identical unit tardiness penalties. For each instance, we drew samples of random integers to obtain values of p_j, d_j, and v_j. For values of p_j we drew random integers from the range [1,99]. For values of d_j we followed the method used in several studies of the total tardiness problem by specifying a tardiness factor and a due date range (Baker and Martin, 1974; Potts and Van Wassenhove, 1988). The tardiness factor represents the proportion of the jobs that will be late in an arbitrary sequence when all jobs have the same due date. This parameter determines the average due date. The due date range represents the range of the due dates as a proportion of the expected makespan (which in this case is $50n$). The test data were created by selecting values of a tardiness factor (TF = 0.25, 0.5, 0.75, 1.0) and a due date factor (DDR = 0.25, 0.5, 0.75) and then sampling the set of due dates. Finally, for job j, we then drew the value of v_j as a sample from a discrete uniform distribution between $0.5p_j$ and $1.5p_j$. We found that this mechanism led to instances in which the optimal proportion of accepted jobs was about 0.7 on average.

For each of the 12 combinations of TF and DDR, we replicated the procedure five times, giving rise to 60 test problems for a given problem size. We then repeated the procedure for n = 10, 15, 20, and 25, for a total of 240 test problems. We solved each problem first with an optimization approach and then with a heuristic approach. Our results are described in the next two sections.

Results for the Optimization Model

Once the formulation is built on a spreadsheet, the optimal solution can be found using RSP and the Gurobi Solver Engine. In our experiments, we terminated the optimization run at 3600 seconds (one hour). Our results are summarized in Table 1.

Table 1. Performance of the Optimizing Procedure

Problem Size (n)	Percent Accepted	Median seconds	Mean seconds	Maximum seconds
10	68%	0.3	0.3	0.7
15	75%	0.4	0.7	8.1
20	73%	0.8	4.7	177.6
25	76%	1.0	199	3600*

As Table 1 indicates, our mechanism for generating random test problems led to instances in which the number of jobs accepted in the optimal solution tended to average a little over 70% of the given job set. This feature seems to be a desirable one compared to the extreme alternatives. For example, if we knew the optimal solution contained no accepted jobs, we could find it with only a trivial effort. At the other extreme, if we knew the optimal solution contained no rejected jobs, we could equivalently solve the corresponding total tardiness problem. That is not a trivial task, but very efficient branch and bound methods have been developed that can handle problems with hundreds of jobs. Between these extremes, intermediate values reflect the combinatorial nature of both job selection and sequencing and therefore ought to be the most challenging. Our method of generating test problems appears to generate challenging instances.

For each problem size, Table 1 shows the median, average, and maximum time required to solve the problem (in seconds), for each set of 60 test problems. None of the 180 instances containing 10, 15, or 20 jobs required as much as three minutes to solve. On the other hand, two of the 25-job test problems could not be solved within the one-hour time limit. But even counting the one-hour run times, the average time to solve a 25-job problem was less than four minutes and the median time was barely above one second. (This is indeed a very skewed distribution of run times, but it is not necessarily unusual for complicated combinatorial problems.) The longest run times occurred for larger values of tardiness factor and due date range, although that was not the case in every random instance.

We tested the alternative MIP formulations mentioned earlier on each of the 25-job problems. It turned out that each formulation encountered at least one instance that could not be solved in one hour of run time. However, no instance required an hour of run time by all of the formulations. In other words, all of the test problems were solvable within the one-hour time limit by at least one of the formulations.

We conclude that an MIP approach easily handles problems of up to 20 jobs and is likely to handle larger problems as well, most of the time. This finding is impressive in comparison with the results for general versions of the problem, which we summarized earlier.

Nobibon and Leus (2011) solved the OAS problem with weighted tardiness assuming that a certain subset of the jobs corresponds to firm planned orders (i.e., jobs that cannot be rejected). In the special case of an empty set of firm planned orders, their model is identical to that of Slotnick and Morton, and their computational results are more recent. However, most of their test problems were based on sampling processing times between 1 and 10. Their exploration of sampling between 10 and 100 showed that “large processing times” can lead to much longer run times. Nevertheless, keeping in mind that their objective function contains weights, it is interesting to note that for 10-job instances with large processing times (a case that corresponds most closely to ours), their most promising solution algorithm required as much as 40 seconds to produce a solution.

Heuristic Approach

For problems with more than 25 jobs, we would be interested in the use of heuristic methods to solve the OAS problem with tardiness penalties because an MIP solution may require prohibitive

amounts of computational effort. Fortunately, RSP offers an effective heuristic algorithm as well as an optimizing algorithm. In particular, we can find solutions using the so-called Standard Evolutionary Engine (SEE), which is a flexible but proprietary genetic algorithm available as an option to users of RSP. The SEE has previously been shown to perform well in solving sequencing problems (Baker and Camm, 2005; Baker, 2011). We might expect it to accommodate job acceptance decisions in addition to sequencing decisions and still perform well. In order to test the effectiveness of the SEE, we used it to solve all 240 test problems, giving us a basis for evaluating the effectiveness of a heuristic approach.

In fact, alternative formulations of the OAS problem with tardiness penalties are possible with the SEE. The most intuitive approach might be to make the accept/reject decision explicit (with a binary variable) and combine that with a traditional sequencing model. Figure 1 shows a corresponding spreadsheet layout for a problem containing 10 jobs. Rows 3-6 display the parameters of the problem. Rows 8 and 9 provide the decision variables: row 8 shows the job sequence and row 9 shows the binary decisions for accept (1) or reject (0). Given this information, it is straightforward to make the necessary calculations to determine the corresponding objective function. In the lower portion of the table, we record the processing time, due date and revenue value for each of the jobs in the given sequence. We also calculate each job's completion time, tardiness, and profit (if accepted). The sum of the numbers in row 17 appears in cell A18. This figure is the objective function.

	A	B	C	D	E	F	G	H	I	J	K
1	**Heuristic 1**										
2											
3	Job	1	2	3	4	5	6	7	8	9	10
4	Ptime	59	11	57	92	38	98	93	91	36	10
5	Ddate	310	194	216	279	261	301	272	218	226	274
6	Value	31	6	32	94	42	147	95	59	40	7
7											
8	Sequence	5	1	3	9	7	4	2	10	8	6
9	Accept	1	0	0	1	1	1	0	0	0	1
10											
11											
12	Ptime	38	59	57	36	93	92	11	10	91	98
13	Complete	38	38	38	74	167	259	259	259	259	357
14	Ddate	261	310	216	226	272	279	194	274	218	301
15	Tardiness	0	0	0	0	0	0	65	0	41	56
16	Value	42	31	32	40	95	94	6	7	59	147
17	Profit*Acc	42	0	0	40	95	94	0	0	0	91
18	362										
19											

Figure 1. Spreadsheet layout for Heuristic 1.

For the purposes of using the SEE, the problem is formulated as follows.

Objective:	A18 (maximize)
Decisions:	B8:K9
Constraints:	B8:K8 = alldifferent
	B9:K9 = binary

Thus, the spreadsheet model has 2*n* variables and 2*n* constraints. A solution is found by implementing the SEE.

Because the SEE is quite flexible and permits the use of complicated spreadsheet functions (as we might infer from Figure 1), other formulations are plausible. Figure 2 shows a simpler formulation. In the spreadsheet of Figure 2, the decision variables correspond to a sequence in row 8. The calculations in the lower portion of the spreadsheet are essentially the same as in Figure 1 but with one exception. In row 17, the profit is replaced by zero if it is negative. This calculation provides the solution algorithm with an incentive to place such jobs toward the end of the sequence (and treat them as rejected), but no explicit acceptance decision is included in the decision variables. Here, the problem is formulated as follows.

Objective:	A18 (maximize)
Decisions:	B8:K8
Constraints:	B8:K8 = alldifferent

Thus, the spreadsheet model has *n* variables and *n* constraints. With fewer variables and constraints than in Figure 1, the ESS might be more effective on this model, even though the accept/reject decision is implicit.

	A	B	C	D	E	F	G	H	I	J	K
1	**Heuristic 2**										
2											
3	Job	1	2	3	4	5	6	7	8	9	10
4	Ptime	59	11	57	92	38	98	93	91	36	10
5	Ddate	310	194	216	279	261	301	272	218	226	274
6	Value	31	6	32	94	42	147	95	59	40	7
7											
8	Sequence	9	4	5	7	6	8	10	3	2	1
9											
10											
11											
12	Ptime	36	92	38	93	98	91	10	57	11	59
13	Complete	36	128	166	259	357	448	458	515	526	585
14	Ddate	226	279	261	272	301	218	274	216	194	310
15	Tardiness	0	0	0	0	56	230	184	299	332	275
16	Value	40	94	42	95	147	59	7	32	6	31
17	Profit*Acc	40	94	42	95	91	0	0	0	0	0
18	362										
19											

Figure 2. Spreadsheet layout for Heuristic 2.

A third formulation attempts to exploit the power of the alldifferent constraint in a different way. For an *n*-job problem, we add *n* additional "dummy" jobs with positive processing times (50, in this example) but with due dates and values of zero. Clearly, there is no incentive to place these jobs near the front of the sequence because they provide no benefit. However, if the SEE places any of the original jobs after one of these dummy jobs, we can conclude that such jobs should be rejected. Stated another way, we need only look at the first *n* sequence positions to identify accepted jobs. Figure 3 shows the relevant portion of the spreadsheet layout. In the solution shown, jobs 1-3 are rejected and do not appear in the first 10 sequence positions. Jobs 8 and 10 do appear, but they are so late that they do not contribute to total profit and are therefore rejected. The problem is formulated as follows.

Objective:	A18 (maximize)
Decisions:	B8:U8
Constraints:	B8:U8 = alldifferent

Thus, the spreadsheet model has 2*n* variables and 2*n* constraints.

	A	B	C	D	E	F	G	H	I	J	K	L
1	**Heuristic 3**											
2												
3	Job	1	2	3	4	5	6	7	8	9	10	11
4	Ptime	59	11	57	92	38	98	93	91	36	10	50
5	Ddate	310	194	216	279	261	301	272	218	226	274	0
6	Value	31	6	32	94	42	147	95	59	40	7	0
7												
8	Sequence	4	9	7	5	6	8	10	13	14	15	11
9												
10	10											
11												
12	Ptime	92	36	93	38	98	91	10	50	50	50	50
13	Complete	92	128	221	259	357	448	458	508	558	608	658
14	Ddate	279	226	272	261	301	218	274	0	0	0	0
15	Tardiness	0	0	0	0	56	230	184	508	558	608	658
16	Value	94	40	95	42	147	59	7	0	0	0	0
17	Profit*Acc	94	40	95	42	91	0	0	0	0	0	0
18	362											
19												

Figure 3. Spreadsheet layout for Heuristic 3.

Results for the Heuristic Procedures

Implementing the SEE on a spreadsheet involves an effort similar to that of implementing an optimization algorithm. First, the model is built on the spreadsheet, as in Figures 1-3. Next, RSP is invoked with the specification of objective, decision variables, and constraints. Then, certain control parameters for the SEE are specified, and finally the algorithm is executed. Normally, for the solution of a single problem, the user would run the SEE several times in succession, modifying its control parameters in light of results encountered in previous runs, until the search seems to have exhausted reasonable combinations of parameters. For the purposes of a computational study, however, manual implementation is unwieldy, so we substituted a restrictive implementation that lends itself to automated experiments. The steps in this automated implementation are shown in Figure 4.

Automated Implementation of the SEE

1. Initialize the sequence according to job number and assume all jobs are accepted.
2. Run the SEE using its default parameters.

3. Re-run the SEE to see whether its randomized elements produce an improvement. If so, continue to re-run the algorithm.
4. When a run does not improve the solution, re-initialize the sequence starting with the reverse of the initial ordering, increase the mutation rate, and run the SEE.
5. Repeat Steps 2 and 3.
6. Select the better of the solutions generated from the two initialization procedures.

Figure 4. SEE Implementation in the Experiments

The SEE contains some randomized steps (specifically, in the generation of mutations). As a consequence, two successive runs of the SEE under the same parametric conditions may produce different results. The automated implementation in Figure 4 does not fully exploit the effects of randomization. In fact, a manual implementation could probe for such effects. Moreover, as mentioned earlier, a manual implementation could tailor parametric settings to the outcomes of previous runs, a feature missing from the automated implementation. Therefore, it seems reasonable to claim that the performance of the SEE in our computational experiments is likely to underestimate its potential to produce good solutions.

One of the key parameters associated with the SEE is the run time: the user typically sets the maximum length of time the algorithm will run before being terminated. (Other parametric settings may induce earlier termination.) In these experiments, the time limit on each run was set at five seconds. (The implementation described in Figure 4 calls for at least four runs of the SEE per problem.) The time limit could easily be set to a higher number; this is another reason why the potential performance of the SEE may be underestimated in these experimental results.

For each problem, we calculate the ratio of the heuristic solution to the optimal solution. When the heuristic produces an optimum, this ratio is equal to 1; otherwise, it is below 1. Table 2 summarizes the values of these ratios in the 240 test problems. For each heuristic, the table shows the minimum (or worst case) ratio in the 60 test problems of a given size, the mean ratio in the 60 test problems, and the number of times (out of 60) that an optimum was produced. Overall, Heuristic 2 produced an optimum in 84% of the test problems, Heuristic 1 in 75%, and Heuristic 3 in 53%.

Problem	Heuristic 1			Heuristic 2			Heuristic 3		
Size (n)	Min	Mean	Optima	Min	Mean	Optima	Min	Mean	Optima
10	0.953	0.999	58	0.998	1.000	59	0.953	0.998	52
15	0.908	0.988	33	0.941	0.998	47	0.903	0.980	17
20	0.922	0.998	46	0.924	0.998	52	0.965	0.996	34
25	0.927	0.996	42	0.927	0.997	44	0.926	0.993	25

Table 2. Results for the Three Heuristic Procedures

The mean values reflect the same ordering: Heuristic 2 produces the best mean ratio, just slightly better than Heuristic 1, while Heuristic 3 is not quite as effective. Nevertheless, it is remarkable that these heuristic solutions are optimal in a majority of cases, given that the OAS problem with identical unit tardiness penalties is an NP-Hard problem to begin with. Such good performance reflects the power of the SEE, especially on sequencing problems (or, in this case, on a variation of a pure sequencing problem). In particular, Heuristic 2, which is the best of the three approaches by a narrow margin, achieves an objective function that misses the optimum by an average of about 0.2%. Given that the corresponding spreadsheet model is easy to build and easy to implement with RSP, this represents excellent performance. (For comparison, Rom and Slotnick (2009) designed a genetic algorithm for the more general OAS problem and were able to achieve suboptimality levels of about 0.6%.)

Summary and Conclusions

We have focused on the OAS problem with identical unit tardiness penalties and positioned it alongside previous research on order acceptance and scheduling and research on scheduling with job rejection. We first proposed an optimization approach to solving the problem, relying on a spreadsheet-based MIP model suitable for solution using RSP. We found that the optimization approach is practicable for problem sizes up to 25, whereas other versions of the OAS problem have been difficult to solve for problem sizes as small as 10.

We built three spreadsheet-based formulations of the model suited to heuristic solution using the Standard Evolutionary Engine in RSP. All three of these heuristic approaches performed well in the computational study, producing solutions that on average lie well within 1% of the optimum. The approaches shown in Figures 1 and 2 exhibited better performance than the approach shown in Figure 3, although no single heuristic approach dominated the other two in every test problem. The use of RSP obtained optimal solutions in a majority of test problems and delivered suboptimal solutions on the other occasions, averaging less than 0.2% deviation from optimality.

The novelty of this material lies in the mixed-integer program, the models underlying the heuristic procedures, and the computational comparisons. The optimization model and heuristic models represent benchmarks for any future study of the OAS problem with tardiness penalties. As mentioned earlier, RSP is an accessible piece of software, and it is already familiar to a wide audience. (Frontline Systems reports more than 250,000 downloads of RSP since 2009.) Thus the computing tools used in this study are well known. However, our ability to make progress with these tools is impressive compared to analogous results elsewhere in the OAS literature. Therefore, another contribution of this work is to reinforce the use of RSP as a computationally advanced software package suitable for combinatorial problems in scheduling.

References

Baker, K. (2011). Solving sequencing problems in spreadsheets. *International Journal of Planning and Scheduling* 1 (2011), 3-18.

Baker, K. and J. Camm (2005). On the use of integer programming vs. evolutionary solver in spreadsheet optimization. *INFORMS Transactions on Education* **5**, 1-7.

Baker, K. and B. Keller (2010). Solving the single-machine tardiness problem using integer programming. *Computers & Industrial Engineering* 59, 730-735.

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.

Bartal, Y., S. Leonardi, A.Marchetti-Spaccamela, J. Sgall, and L. Stougie (2000). Multiprocessor scheduling with rejection. *SIAM Journal of Discrete Mathematics* 13, 64-78.

Cesaret, B., C. Oguz, and F. Salman (2012) A tabu search algorithm for order acceptance and scheduling. *Computers & Operations Research* 39, 1197-1205.

Charnsirisakskul , K., P. Griffin, and P. Keskinocak (2004). Order selection and scheduling with leadtime flexibility. *IIE Transactions* 36, 697-707.

Du, J. and J. Leung (1990). Minimizing total tardiness on one machine is NP-hard. *Mathematics of Operations Research* 15, 483-495.

Keha, K., K. Khowala, and J.Fowler (2009). Mixed integer programming formulations for single machine scheduling problems. *Computers & Industrial Engineering* 56, 357-367.

Nobibon, F. and R. Leus (2011). Exact algorithms for a generalization of the order acceptance and scheduling problem in a single-machine environment. *Computers & Operations Research* 38, 367-378.

Oguz, C., S. Salman, and Z.Yalcin, (2010). Order acceptance and scheduling decisions in make-to-order systems. *International Journal of Production Economics* 125, 200-211.

Potts, C. and L. Van Wassenhove (1988). Algorithms for scheduling a single machine to minimize the weighted number of late jobs. *Management Science* 34, 843-858.

Rom, W. and S. Slotnick (2009). Order acceptance using genetic algorithms. *Computers & Operations Research* 36, 1758-1767.

Roundy, R., D. Chen, P. Chen, M. Cakanyildirim, M. Freimer, and V. Melkonian (2005). Capacity-driven acceptance of customer orders for a multi-stage batch manufacturing system: models and algorithms. *IIE Transactions* 37, 1093-1105.

Slotnick, S. (2011). Order acceptance and scheduling: a taxonomy and review. *European Journal of Operational Research* 212, 1-11.

Slotnick, S. and T. Morton (1996). Selecting jobs for a heavily loaded shop with lateness penalties. *Computers & Operations Research* 23, 131-40.

Slotnick, S. and T. Morton (2007). Order acceptance with weighted tardiness. *Computers & Operations Research* 34, 3029-3042.

Yang, B. and J. Geunes (2007). A single resource scheduling problem with job-selection flexibility, tardiness costs and controllable processing times. *Computers & Industrial Engineering* 53, 420-432.

Zhang, L., L. Lu, and J. Yuan (2009). Single machine scheduling with release dates and rejection. *European Journal of Operational Research* 198, 975-978.