SOLVING THE SINGLE-MACHINE SEQUENCING PROBLEM USING INTEGER PROGRAMMING

Kenneth R. Baker
Tuck School
Dartmouth College
Hanover, NH 03755
ken.baker@dartmouth.edu
Ph. 603-646-2064
Fx. 603-646-1308

Brian Keller
Department of Systems and Industrial Engineering
University of Arizona
Tucson, AZ 85721
brndnlkllr@yahoo.com

ABSTRACT

Various integer programming models have been proposed for sequencing problems. However, little is known about the practical value of these models. This paper reports a comparison of six different integer programming formulations of the single-machine total tardiness problem. We created a set of especially difficult test problems and attempted to solve them with each of the formulations, using CPLEX software. We found that one formulation performs much more effectively than the others. A generic integer programming approach is still not capable of solving problems with hundreds of jobs, so in that respect, it does not compete with state-of-the-art tardiness algorithms. However, the integer programming approach remains viable for problems containing as many as 40 or 50 jobs and may be the better algorithmic choice when convenience in implementation is considered.

SOLVING THE SINGLE-MACHINE TARDINESS PROBLEM USING INTEGER PROGRAMMING

ABSTRACT

Various integer programming models have been proposed for sequencing problems. However, little is known about the practical value of these models. This paper reports a comparison of six different integer programming formulations of the single-machine total tardiness problem. We created a set of especially difficult test problems and attempted to solve them with each of the formulations, using CPLEX software. We found that one formulation performs much more effectively than the others. A generic integer programming approach is still not capable of solving problems with hundreds of jobs, so in that respect, it does not compete with state-of-the-art tardiness algorithms. However, the integer programming approach remains viable for problems containing as many as 40 or 50 jobs and may be the better algorithmic choice when convenience in implementation is considered.

INTRODUCTION

Imagine that you are a member of the Operations Research staff in industry and that you have been given the task of finding optimal solutions to an important sequencing problem. Suppose further that the problem matches a well-known model that appears in the research literature, such as the total tardiness problem. How would you approach the task?

Assuming that you are familiar with the literature on this topic, you are probably aware that academic researchers have developed specialized algorithms capable of solving relatively large versions of the tardiness problem, for hundreds of jobs (Szwarc et al., 2001). One approach, therefore, is to develop code to implement a state-of-the-art algorithm for the specialized purpose at hand. (Such algorithms first appeared in the literature around 1968, and major improvements have been developed in every decade since.) However, this could be a complicated and time-consuming task. The specialized nature of the algorithm may take quite a bit of time to master, and its application may require some testing and tweaking beyond the published description. In a practical setting, the potential for fast solution times using a specialized algorithm must ultimately be balanced against ease of implementation.

An alternative approach is to use a more general modeling capability—in this case, integer programming (IP). By now, it is fairly common for Operations Research staff members to have access to a commercial code for IP (such as CPLEX) and moreover, to have some experience in applying it. Could you be as effective in finding solutions using an IP approach as you would be with a complicated and specialized algorithm?

This question is difficult to answer for two reasons. First, although several alternative approaches exist to the IP formulation of scheduling problems, little comparative testing of those formulations has been done. We can count the number of variables and constraints in a formulation, and sometimes we can evaluate the tightness of lower bounds, but ultimately we need to investigate computation times to determine which IP approach is most suitable. Second, we don't have much information about the capabilities of commercial optimization packages at solving standard sequencing problems. In contrast to the highly-tailored

algorithms that have been proposed and tested in the research literature, we lack a computational perspective on the use of "vanilla" IP software. In this paper, we address these two topics and provide some insight into the capabilities of an IP approach.

We focus on the single-machine tardiness problem, which is perhaps the most familiar basic problem in sequencing. In the tardiness problem, job j is characterized by a processing time (p_j) and a due date (d_j). A feasible sequence determines the job's completion time (C_j). Based on the set of completion times generated by a particular sequence, we can calculate the value of total tardiness:

$$\text{Total tardiness} = z = \sum_{j=1}^{n} \max\{0, C_j - d_j\}$$

Our experiments involve solving the single-machine tardiness problem using IP. After introducing the model and tracing the historical development of IP approaches, we describe our experimental design, summarize our results, and draw conclusions for the use of IP in solving sequencing problems.

LITERATURE REVIEW

The first research articles describing how to solve sequencing problems with IP date back to 1959. (Actually, the earliest papers addressed multi-machine sequencing problems, but their formulation strategies apply as well to the single-machine model.) At the time, IP algorithms were just being developed, and sequencing problems were viewed as a convenient vehicle for exploring the computational capability of new algorithms. A variety of articles explored clever ways of modeling the sequencing problem using IP.

The first publication on the topic is due to Wagner (1959), whose work focused on minimizing makespan in the three-machine flow shop problem. In Wagner's formulation, the key binary variables are

x_{ik} = 1 if job i is assigned to the kth position in sequence

We refer to this as the *sequence-position variable*. Building around this definition, it is not difficult to develop an IP model for the single-machine tardiness problem. A generalization to weighted tardiness is possible but not straightforward. Thus, a formulation that relies on these variables may not be as flexible as other formulations.

Another early paper on the topic is due to Bowman (1959), whose work focused on minimizing makespan in the job shop problem. In Bowman's formulation, the key binary variables are

x_{jt} = 1 if job j is in process during period t

This definition is essentially equivalent to a slightly different version that is now typically referred to as *time indexing*. Assuming that time is discrete, any job is either in process or is not, during any time period.

Yet another early paper on the topic is due to Manne (1960), who was also focused on the makespan problem for the job shop. In Manne's formulation, the key binary variables are

y_{ij} = 1 if job j follows job i

We refer to this as a *precedence variable*. For any pair of jobs i and j, either j follows i or i follows j. If we let s_j denote the start time of job j, then either $s_i + p_i \leq s_j$ or $s_j + p_j \leq s_i$. These are called *disjunctive constraints*, meaning that one or the other must hold for a solution to be feasible. Using precedence variables, the pair of disjunctive constraints between jobs i and j can be written in linear form as follows:

$$s_i + p_i \leq s_j + M(1 - y_{ij})$$
$$s_j + p_j \leq s_i + My_{ij}$$

Here, M denotes a large positive constant.

As mentioned above, early papers were primarily concerned with IP formulations and the capability of IP algorithms. The authors viewed sequencing problems as a means to test the practical capabilities of algorithms rather than as research targets themselves. In a similar fashion, the traveling salesperson problem (TSP) was also often used as a means of testing general ideas of combinatorial optimization, and in the case of IP, a paper due to Miller et al. (1960) introduced the *index method* for formulating TSPs. The key requirement for a TSP formulation is to enforce tour constraints, meaning that the sequence must form a full tour of the given elements (cities, in the TSP). In the sequencing problem, all the given elements must be sequenced, although the tour need not return to its starting point. The index method relies on two types of binary variables,

y_{ij} = 1 if job i is scheduled before j in sequence
u_{ij} = 1 if job i is scheduled immediately before j in sequence

The y_{ij} variables are precedence variables, as introduced earlier, and the u_{ij} variables are called *immediate-precedence variables*. We elaborate later on the index method.

Another important paper appeared near the end of the 1960's. Pritsker et al. (1969) described an IP approach to the project scheduling problem, and in the process introduced the following time-indexed variables.

x_{jt} = 1 if job j completes in period t

Just as easily, we could define x_{jt} = 1 if job j starts in period t, which is the more common treatment of time-indexed variables. These variables are slightly different from Bowman's, but the concept is similar: from the fact that x_{jt} = 1, we can deduce other consequences, including the impact of job j in the objective function. Also similar is the need to impose constraints that ensure consistency among the time-indexed variables.

None of the specific papers mentioned in this review actually dealt with the single-machine tardiness problem, but their ideas all contributed to alternative IP formulations. However, in the years that followed, virtually all the computational work on solving single-machine sequencing problems emphasized combinatorial optimization techniques tailored to specific sequencing models. Few researchers have revisited the capability of IP in light of the many hardware and software advances that have been made. We are aware of only two

exceptions. Khowala et al. (2005) studied the weighted tardiness problem with CPLEX 8.1 but failed to find optimal solutions for most of their test problems. Tseng et al. (2004) studied relatively small versions of the flow shop makespan problem. Their work was subsequently extended to larger problems in Stafford et al. (2005). The present paper is actually the first to explore the use of IP for solving the tardiness problem.

FORMULATIONS OF THE TARDINESS PROBLEM

This section reviews six alternative IP formulations. The processing times p_j and the due dates d_j are considered to be given parameters. The principal variables were introduced in the previous section, but we define them for each formulation, as needed.

Formulation DJ (Disjunctive constraints)

In Manne's approach, as indicated earlier, the key binary variable is $y_{ij} = 1$ if job j follows job i, and zero otherwise. It is sufficient to work with these variables for $i < j$. We know that

$$y_{ji} = 1 - y_{ij}$$

However, this disjunctive relationship is implicit in the disjunctive constraints that define the start times s_j. To construct the objective function, we also define t_j as the tardiness for job j. We track the tardiness value by introducing the following constraint.

$$s_j + p_j - d_j \le t_j$$

Because t_j is nonnegative, the constraint ensures that $t_j = \max\{0, s_j + p_j - d_j\}$

Variables

$y_{ij} = 1$ if job j follows job i, and zero otherwise ($i < j$)
s_j = start time of job j
t_j = tardiness of job j

Constraints

$s_i + p_i \le s_j + M(1 - y_{ij})$, for all job pairs (i, j) with $i < j$
$s_j + p_j \le s_i + My_{ij}$, for all job pairs (i, j) with $i < j$
$s_j + p_j - d_j \le t_j$, for all jobs j

Objective

$$\text{Minimize } \sum_{j=1}^{n} t_j$$

Formulation SP (Sequence position)

In this formulation, we rely on the sequence-position variables originally proposed by Wagner.

Variables

x_{ik} = 1 if job i is kth in sequence
t_k = tardiness of kth job in sequence

Constraints

$$\sum_{i=1}^{n} x_{ik} = 1, \quad \text{for all positions } k$$

$$\sum_{k=1}^{n} x_{ik} = 1, \quad \text{for all jobs } i$$

$$\sum_{i=1}^{n} p_i \sum_{u=1}^{k} x_{i\ u} - \sum_{i=1}^{n} d_i x_{i\ k} \le t_k, \quad \text{for all positions } k$$

Objective

$$\text{Minimize } \sum_{k=1}^{n} t_k$$

Formulation LO *(linear ordering variables)*

The notion of linear ordering variables arises in other types of combinatorial problems. Its first appearance in a paper on scheduling appears to be due to Nemhauser & Savelsbergh (1992), who examined a model with release dates as a basis for developing a specialized cutting plane algorithm. The formulation uses precedence variables.

Variables

y_{ij} = 1 if job i is scheduled before j in sequence ($i \neq j$)
t_j = tardiness of job j

Constraints

$y_{ij} + y_{ji} = 1$, for all jobs pairs (i, j) $i \neq j$
$y_{ij} + y_{jk} + y_{ki} \le 2$, for all job triplets (i, j, k) with $i \neq j$, $i \neq k$, and $j \neq k$

$$p_j + \sum_{i=1}^{n} p_i y_{ij} - d_j \le t_j, \quad \text{for all jobs } j$$

Objective

$$\text{Minimize } \sum_{j=1}^{n} t_j$$

Formulation HY (Hybrid)

We construct a hybrid formulation by using both the sequence-position variables and the precedence variables. A set of constraints is needed to ensure that the two sets of variables are consistent. Again, t_j denotes the tardiness of job j.

Variables

x_{ik} = 1 if job i is kth in sequence

y_{ij} = 1 if job i is scheduled before j in sequence ($i < j$)
t_j = tardiness of job j

Constraints

$$\sum_{i=1}^{n} x_{ik} = 1, \quad \text{for all positions } k$$

$$\sum_{k=1}^{n} x_{ik} = 1, \quad \text{for all jobs } i$$

$$x_{j\,k} + \sum_{u=1}^{k-1} x_{i\,u} \le 1 + y_{i\,j}, \quad \text{for all } k, \text{ all } j > 1, \text{ and all } i < j$$

$$p_j + \sum_{i=1}^{n} p_i y_{ij} - d_j \le t_j, \quad \text{for all jobs } j$$

Objective

$$\text{Minimize } \sum_{j=1}^{n} t_j$$

Formulation TR (Tour constraints)

As mentioned earlier, the similarity of the sequencing problem and the TSP allows us to adapt a different IP approach. We augment the problem by adding a job 0 with $p_0 = 0$, and we use precedence variables y_{ij} and immediate precedence variables u_{ij}. The completion time of job j can then be written as $\sum_{i=1}^{n} p_i y_{ij} + p_j$. In addition, $\sum_{i=1}^{n} y_{ij}$ represents the number of jobs preceding job j, and $\sum_{k=1}^{n} y_{jk}$ represents the number following. The constraints form a tour for the (n + 1) jobs.

Variables

y_{ij} = 1 if job i is scheduled before j in sequence ($i \neq j$)
u_{ij} = 1 if job i is scheduled immediately before j in sequence ($i \neq j$)
t_j = tardiness of job j

Constraints

$$\sum_{i=0}^{n} u_{ij} = 1, \quad \text{for all jobs } j \ge 0$$

$$\sum_{i=0}^{n} u_{ji} = 1, \quad \text{for all jobs } j \ge 1$$

$$\sum_{i=1}^{n} y_{ij} + \sum_{k=1}^{n} y_{jk} = n - 1, \quad \text{for all jobs } j \ge 1$$

$$\sum_{i=1}^{n} p_i y_{i\,j} + p_j - d_j \le t_j, \quad \text{for all jobs } j \ge 1$$

$$\sum_{i=1}^{n} y_{i\ j} - \sum_{i=1}^{n} y_{i\ k} + (n+1)u_{j\ k} \le n, \quad \text{for job pairs } (j,k) \text{ with } j > 0,\ k > 0,\ j \ne k$$

As introduced by Desrochers and Laporte (1991), a tighter constraint can be substituted for this last one. (We used the tighter constraint in the computational experiments.)

$$\sum_{i=1}^{n} y_{ij} - \sum_{i=1}^{n} y_{ik} + (n+1)u_{jk} + (n-1)u_{kj} \le n, \quad \text{for job pairs } (j,k) \text{ with } j > 0,\ k > 0,\ j \ne k$$

Objective

$$\text{Minimize } \sum_{j=1}^{n} t_j$$

Formulation TI (Time-indexed model)

In the time-indexed model, we view time as discrete and formulate the model in terms of time periods. In the case of the single-machine sequencing model, we know in advance that the makespan will be $T = \sum_{j=1}^{n} p_j$. The parameter T is an important part of the model and is discussed below. We also let S_{jt} denote the set of start times for which job j could be in process during period t.

$$S_{jt} = \{\max(1,\ t - p_j + 1),\ \ldots,\ \min(t,\ T - p_j + 1)\}$$

Variables

$x_{jt} = 1$ if job j starts in period t (i.e. at the beginning of period t)

Constraints

$$\sum_{t=1}^{T-p_j+1} x_{jt} = 1, \quad \text{for all jobs } j$$

$$\sum_{j=1}^{n} \sum_{s \in S_{jt}} x_{js} \le 1, \quad \text{for all time periods } t = 1, 2, \ldots, T$$

Objective

$$\text{Minimize } \sum_{j=1}^{n} \sum_{t=1}^{T-p_j+1} c_{jt} x_{jt}$$

In the objective function, c_{jt} denotes the tardiness of job j if it begins processing in period t. That is, $c_{jt} = \max\{0,\ (t + p_j - 1 - d_j)\}$.

Unlike the other formulations, the size of the time-indexed formulation depends on the total processing time. Suppose we assume that an average processing time is 50. (This value reflects the properties of our test data, but it still seems reasonable in general if we want to discriminate among processing times on a scale of roughly 1 to 100.) Then, when the

problem contains 20 jobs, the makespan will be about $T = 1000$. A 20-job problem would therefore lead to a formulation containing 1020 constraints and 20,000 variables, all of which are binary. Obviously, the time-indexed formulation can lead to models with a very large number of variables; however, it may be efficient in other respects. Dyer and Wolsey (1990), who address a version of the sequencing problem with release dates, show that lower bounds obtained from the time-indexed formulation are quite tight. Pan and Shi (2006) exploit the lower bounds of the time-indexed formulation in their algorithm for the weighted tardiness problem. However, no articles investigate the computational properties of solutions obtained for the tardiness problem by the time-indexed IP model or comparisons to other formulations.

TEST PROBLEMS AND EXPERIMENTS

To evaluate the capability of an IP approach, it should be tested on difficult problem instances. Test conditions that lead to near-trivial problems should be avoided, and we should attempt to focus on conditions under which the solution procedure is most severely challenged (Hall and Posner, 2001). Fortunately, previous research gives us some insight into how to construct tardiness problems that are difficult to solve.

In our test problems, we first sampled processing times as integers uniformly distributed between 1 and 100. Thus, the average processing time was about 50. This aspect of the design, which may seem innocuous in other settings, is important because of its effect on the time-indexed formulation. As mentioned above, the number of variables in Formulation TI can become quite large. If we had sampled processing times as integers between 1 and 10, the performance of the TI formulation would undoubtedly look better.

We chose the range between 1 and 100 for two reasons. The first of these is historical consistency. Most of the computational tests involving scheduling algorithms sample processing times from a uniform distribution on [1, 100] or a normal distribution with a mean of 100 and a standard deviation as large as 25. The latest example in the tardiness literature is Szwarc et al. (2001), which represents the state-of-the-art paper on specialized algorithms for the tardiness problem. Exceptions to this approach tend to be authors presenting a new algorithm which, like the time-indexed model, is favored by small processing times.

The second reason is perhaps more philosophic. Ideally, we would like to use data that are representative of "real" scheduling problems, but such data are scarce. However, we believe that a range of 100 possibilities is reasonable when there are as many as 40 or 50 elements in the sample. Sampling between 1 and 10 would create many ties, which would not necessarily be realistic. In fact, we believe that the very notion of *problem size* intends to count the number of different elements in the problem. Moreover, in the case of most specialized algorithms, the presence of ties tends to make optimal solutions easier to find (Potts and Van Wassenhove, 1982).

Test problems for the total tardiness objective have been used in many research investigations, dating back to Srinivasan (1971) and to Wilkerson and Irwin (1971). By now, it is widely accepted that certain numerical features make a tardiness problem relatively easy or difficult. Two insights support this notion. First, problems are most difficult to solve when some, but not all, of the jobs are likely to be tardy. Stated another way, problems are most difficult when the due dates, on average, lie neither at the beginning nor at the end of the schedule. Second, if the due dates are dispersed too widely, then it may be relatively easy to

place the jobs on a time scale so individual due dates can be met. Thus, problems are most difficult to solve when due dates exhibit a limited amount of dispersion.

In this spirit, define the *tardiness factor*, denoted *TF*, as the fraction of the jobs likely to be tardy. The tardiness factor is usually a parameter of the data-generating process. Let μ_p denote the mean of the distribution from which processing times are sampled and let μ_d denote the mean of the distribution from which due dates are sampled. Then:

$$TF = 1 - \mu_d / (n\mu_p)$$

For a desired level of *TF*, we thus have:

$$\mu_d = n\mu_p(1 - TF)$$

Next, define the *due-date range*, denoted *DDR*, as the range of the due dates relative to the expected total processing time. In the process of generating test data, we sample due dates from a uniform distribution on the interval (*a*, *b*). This implies

$$DDR = (b - a) / (n\mu_p)$$

Thus, for a desired value *DDR*, we calculate the width of the range, $(b - a) = n\mu_p(DDR)$.

To construct test problems, we first decide on a processing time distribution and choose its mean. Next, for a given *TF*, we calculate the mean of the due date distribution, and for a given *DDR*, we determine the width of the due date distribution. Difficult problems are often associated with high tardiness factors,[1] so we used *TF* = 0.5 to 0.8 in steps of 0.1. Difficult problems are also associated with tight due date ranges[2], so we used *DDR* = 0.15 to 0.25 in steps of 0.05. These choices give rise to 12 different parametric combinations, and we generated 5 test problems for each combination, or 60 test problems for each value of *n*.

Our experiments involved solving each test problem using CPLEX 11.0 on a 900 MHz UltraSPARC-III with 4 GB of memory. We imposed a 3600-second time limit and simply terminated a particular run if the optimal solution had not been found and verified in that amount of time.

COMPUTATIONAL RESULTS

Our first experiments used test problems containing 10 jobs. Even at this problem size, we found that the TR formulation required excessive amounts of time and seldom produced optimal solutions within the time limit. The runs are summarized in Table 1. For each formulation, the table displays the number of problems solved within the time limit, the average time required (in seconds), and the maximum time required. (The average time was calculated over the solved problems, so it is biased downward as an average for the full set of 60 problems.)

We also kept track of the optimality gap for the cases in which an optimal solution was not found and proven. (The optimality gap is the difference between the upper and lower

[1] See, for example, Srinivasan (1971), Fisher (1976), Potts & Van Wassenhove (1982), and Szwarc (2001). Similar results can be found in papers on the weighted tardiness problem.

[2] See, for example, Fisher (1976), Potts & Van Wassenhove (1982), and Szwarc (2001).

bounds, computed as a ratio to the upper bound.) The average value of this gap is shown in the table, as well as the maximum value encountered.[3]

The results show that the TR formulation is very weak, and although the HY formulation solves most of the test problems, it is also inefficient compared to other formulations. We dropped both of these from consideration and proceeded to larger problems.

At a problem size of 20 jobs, the DJ formulation was unable to solve any of the test problems within the time limit, and the LO formulation solved fewer than a third. (Keep in mind, however, that these are not randomly-selected test problems but rather systematically difficult ones.) The TI formulation solved most of the test problems, but reached the time limit on five occasions. In those cases, the optimality gap was modest.

The next set of experiments used problem sizes of 30, 40, and 50. For a problem size of $n = 30$, the SP formulation remained a viable IP approach, whereas the TI formulation began to encounter some difficulty in finding solutions. As shown in Table 1, the TI formulation solved only 34 of the 30-job test problems (and half as many of the 40-job problems). The SP formulation solved all of the 30-job problems and all but two of the 40-job problems. Excluding the two unsolved problems in the latter set, the SP formulation led to solutions within about a minute of cpu time. At problem sizes of 50 jobs, this figure grew to about three minutes, excluding five unsolved problems. However, it was interesting to observe that the optimality gaps remained small in the unsolved problems. In no instance was the optimality gap for the SP formulation as large as 1%.

Table 2 displays the size of the IP formulations for all six alternatives, for $n = 10$ through 50. The number of time periods corresponds to the parameter T in the TI formulation, based on an average processing time of 50. As the table shows, the SP formulation contains the smallest number of constraints, and the number of its variables compares favorably with the other formulations. The DJ formulation has the smallest number of variables, but it is evidently not a tight enough formulation to take advantage of this property. The LO formulation has a large number of constraints, which seems to account for its inability to solve larger problems. The TI formulation has the largest number of variables by a wide margin. In spite of the fact that the TI formulation is known to be very tight, the size of the formulation eventually accounts for its longer computation times.

For 30-job problems, the SP formulation contains 90 constraints and 930 variables, 900 of which are integers. As the table shows, the TI formulation contains about 1530 constraints and about 45,000 integer variables. (These figures are approximate because the length of the schedule is not precisely $50n$ in every case.) The orders-of-magnitude difference in IP model sizes likely accounts for the difference in solution times, even though the TI formulation may be capable of computing tighter lower bounds.

SUMMARY AND CONCLUSIONS

Our experiments tested six different IP formulations for the single-machine tardiness problem using a state-of-the-art software package, CPLEX11.0. We found that one of the six formulations—the model based on "sequence-position" variables—provides the most computationally effective solutions. The formulation solves nearly all difficult problems up

[3] The calculation of the optimality gap was skipped in instances for which no integer solution was reached. This condition occurred infrequently and only in conjunction with the TI formulation.

to about 40 jobs and the vast majority of 50-job problems. When it fails to prove optimality after an hour of computation time, the optimality gap is still quite small. In terms of the question we posed at the outset, we would probably want to use an IP solution approach (as opposed to implementing a specialized algorithm) if we were facing a practical need to solve a tardiness problem containing up to about 50 jobs. The ability to use standard software, instead of researching, coding, and debugging a specialized algorithm, would seem to be the best alternative in terms of ease of implementation.

For scheduling practitioners, the implication is that standard optimization approaches are able to solve moderate-sized scheduling problems in a reasonable amount of time. Specialized algorithms would seem to be preferable only for larger problems, in which an IP approach might take too long. Although our focus was on the single-machine tardiness problem, another implication of our work is that CPLEX and similar kinds of optimization software might also be the best choice (balancing ease of implementation and solution time) for finding solutions to some other types of scheduling problems with different objectives, even for problem sizes that might arise in practice. We hope that our results will motivate further investigations of this possibility.

For scheduling researchers, our results indicate that it may not be enough to refer to "integer programming" approaches to finding optimal solutions. It is important to examine *which* IP formulation is being considered. This point is especially relevant in research projects where the IP solution is used as a benchmark for comparison with tailored approaches such as branch and bound. Furthermore, the attention paid to time-indexed formulations may be justified if they provide insight into theoretical results, but time-indexed models leave something to be desired when it comes to computational performance.

Finally, we repeat the observation that IP modeling has been available as a tool for attacking scheduling problems for roughly half a century. However, systematic studies of its effectiveness have been lacking, especially in the context of present-day hardware and software capabilities. We believe this subject warrants deeper investigation, and we hope that our research will stimulate more such work.

REFERENCES

Bowman, E.H. (1959) The schedule-sequencing problem. *Operations Research* 7, 621-624.

Desrochers, M. and G. Laporte (1991) Improvements and extensions to the Miller-Tucker-Zemlin subtour elimination constraints. *OR Letters* 10, 27-36.

Dyer, M.E. and L.A. Wolsey (1990) Formulating the single machine sequencing problem with release dates as a mixed integer program. *Discrete Applied Math* 26, 255-270.

Fisher, M.L. (1976) A dual algorithm for the one-machine scheduling problem. *Mathematical Programming* 11, 229-251.

Hall, N.G. and M.E. Posner (2001) Generating Experimental Data for Computational Testing with Machine Scheduling Applications. *Operations Research* 49, 854-865.

Khowala, K., A. Keha, and J. Fowler (2005) A comparison of different formulations for the non-preemptive single machine total weighted tardiness scheduling problem. MISTA Proceedings, 643-651.

Manne, A.S. (1960) On the job shop scheduling problem. *Operations Research* 8, 219-223.

Miller, C.E., A.W. Tucker and R.A. Zemlin (1960) Integer programming formulation of Traveling Salesman Problems *Journal of the ACM* 7, 326-329.

Nemhauser, G.L. and M.W.P. Savelsbergh (1992) A cutting plane algorithm for the single machine scheduling problem with release times. In *Combinatorial Optimization: New Frontiers in Theory and Practice* (M. Akgul, H.W. Hamacher, and S. Tufekci, eds), 63-82, Springer-Verlag.

Pan, Y. and L. Shi (2006) On the equivalence of the max-min transportation lower bound and the time-indexed lower bound for single-machine scheduling problems. *Mathematical Programming* 110, 543-559.

Potts, C.N. and L.N. Van Wassenhove (1982) A decomposition algorithm for the single machine total tardiness problem. *Operations Research Letters* 1, 177-181.

Pritsker, A.A.B., L.J. Watters and P.M.Wolfe (1969) Multi-project scheduling with limited resources: a zero-one programming approach. *Management Science* 16, 93-108.

Srinivasan, V. (1971) A Hybrid Algorithm for the One Machine, Sequence-Independent Scheduling Problem with Tardiness Penalties: A Branch-Bound Solution. *Naval Research Logistics Quarterly* 18, 317-327.

Stafford, E.F., F.T. Tseng, and J.N.D. 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.

Tseng, F.T., E.F. Stafford Jr., and J.N.D. Gupta (2004) An empirical analysis of integer programming formulations for the permutation flowshop. *Omega* 32, 285-293.

Wagner, H.M. (1959) An Integer Programming Model for Machine Scheduling. *Naval Research Logistics Quarterly* 6,131-140.

Wilkerson, L.J. and J.D. Irwin (1971) An improved algorithm for scheduling independent tasks. *AIIE Transactions* 3, 239-245.

Table 1. Summary of Computational Results

Problem Size	*IP Formulation*	*Problems Solved*	*Average Time*	*Maximum Time*	*Average Gap*	*Maximum Gap*
10	**DJ**	60	233.67	2579.72	0.00%	0.00%
10	**SP**	60	0.05	0.19	0.00%	0.00%
10	**LO**	60	2.65	8.73	0.00%	0.00%
10	**HY**	58	484.81	3600+	7.80%	9.70%
10	**TR**	7	1401.4	3600+	35.70%	100.00%
10	**TI**	60	10.13	58.85	0.00%	0.00%
20	**DJ**	0	3600+	3600+	96.80%	100.00%
20	**SP**	60	0.6	6.41	0.00%	0.00%
20	**LO**	19	601.63	3600+	26.30%	100.00%
20	**HY**					
20	**TR**					
20	**TI**	55	549.25	3600+	5.80%	14.70%
30	**DJ**					
30	**SP**	60	4.74	44.83	0.00%	0.00%
30	**LO**					
30	**HY**					
30	**TR**					
30	**TI**	34	971.5	3600+	3.64%	23.49%
40	**SP**	58	76.38	3600+	0.32%	0.46%
50	**SP**	55	166.98	3600+	0.33%	0.53%

Table 2. Formulation sizes for the six alternatives

Time Periods	500	1000	1500	2000	2500	3000
Jobs	10	20	30	40	50	60
DJ						
Constraints	100	400	900	1600	2500	3600
Variables	65	230	495	860	1325	1890
Integer variables	45	190	435	780	1225	1770
SP						
Constraints	30	60	90	120	150	180
Variables	110	420	930	1640	2550	3660
Integer variables	100	400	900	1600	2500	3600
HY						
Constraints	560	4220	13980	32840	63800	109860
Variables	155	610	1365	2420	3775	5430
Integer variables	145	590	1335	2380	3725	5370
LO						
Constraints	820	7240	25260	60880	120100	208920
Variables	100	400	900	1600	2500	3600
Integer variables	90	380	870	1560	2450	3540
TR						
Constraints	131	461	991	1721	2651	3781
Variables	230	860	1890	3320	5150	7380
Integer variables	220	840	1860	3280	5100	7320
TI						
Constraints	510	1020	1530	2040	2550	3060
Variables	5000	20000	45000	80000	125000	180000
Integer variables	5000	20000	45000	80000	125000	180000