Solving Sequencing Problems in Spreadsheets

Kenneth R. Baker

Abstract

Suppose that we wish to solve such sequencing problems as the flow shop scheduling problem or the traveling salesperson problem. This paper reports computational experiments that shed light on how large a problem we might be able to solve, assuming that our work is done using a spreadsheet-based approach and the conventional hardware and software available to today's undergraduate student: a laptop and Risk Solver Platform.

Solving Sequencing Problems in Spreadsheets

1. Introduction

Sequencing problems arise commonly in practice but are notoriously difficult to solve. Over the years, many sequencing problems have been attacked with specialized procedures, yielding solutions to larger and larger instances as computational power has increased and algorithmic sophistication has advanced. For example, the latest techniques allow us to compute optimal solutions to some sequencing problems containing hundreds of tasks. At least, that is the nature of the research frontier.

For the practitioner interested in solving such problems or the student exploring the features of such problems, some questions arise about the state of the art. For example, how do we acquire the latest algorithms for our own use? The algorithms are described in research articles, but they are not necessarily directly available to the average practitioner. A good graduate student might be expected to code such algorithms, but they typically involve some tuning and tweaking that research articles don't describe. Harnessing the latest solution algorithm in a practical context or an educational setting is not necessarily an easy task.

A related question arises. How large a problem do we want to solve in the first place? Just because the latest solution algorithm can solve a problem containing 100 tasks, that doesn't mean we're normally interested in problems that large. We may actually want to solve problems of more modest size, although we still want to avoid very demanding computational efforts.

Just as algorithms have continually been improved, so has the computational power available to practitioners and students. Today's laptops provide more computing power than the advanced machines used by researchers when they tested their state-of-the-art algorithms a decade or so ago. Along with powerful hardware, we also have new software that provides access to generic problem-solving capability that would have been available only to an elite group of professionals in years past. This situation leads to an important practical question: if we need to solve a difficult sequencing problem, what can we accomplish with our own laptop and our own generic software? Perhaps the solution is sitting right on our laps, so to speak, rather than in the depths of the research literature.

This paper reports some computational experiments that shed light on these kinds of questions. We suppose that the goal is to find solutions to such sequencing problems as the flow shop scheduling problem or the traveling salesperson problem. But we assume that those solutions will be obtained using conventional hardware and software available to today's student or a generically-trained operations researcher—that is, a modern laptop and spreadsheet-based optimization software. Thus, rather than acquiring some research code for the flow shop problem, or specialized commercial software that will solve a traveling salesperson problem, the approach is to build a generic model on a spreadsheet and find the optimal sequence using spreadsheet-based optimization software. Our

findings indicate that we are no longer limited to "toy" problems, and we can tackle meaningful problem sizes. In most cases, this generic optimization approach does not achieve all the computational results that specialized algorithms might provide, but in some cases the approach is nearly competitive. We devote a section to each problem type and provide formulation details in an appendix.

1.1. The Hardware

The experiments were run on a ThinkPad laptop machine—in particular, Model T420 (with an Intel Core i7 CPU 2.7 GHz processor). This model was available to students in 2011 and represents the capability that a consultant or student might own.

1.2. The Software

The laptop was loaded with the Risk Solver Platform (RSP) add-in.[1] RSP is an advanced version of the solver that comes with Excel. Nevertheless, its educational version is free to adopters of many textbooks, and it is widely used among undergraduate and graduate students who take courses in Management Science or Optimization Modeling. Although RSP has a broader set of features, this study used only the integer programming capability to find optimal solutions—the Gurobi Solver Engine (GSE). The educational version of RSP imposes a ceiling of 200 on the number of variables in a model, but it is possible to download versions that accommodate larger models, at least on a time-limited basis. The optimization runs were terminated after one hour of cpu time in large problems.

1.3. Heuristic Solutions

In parallel to the literature on algorithms for finding optimal solutions to sequencing problems, another stream of research deals with heuristic procedures for solving these same problems. Heuristic procedures generally take modest amounts of computation time but do not guarantee optimal solutions. In large problems, it may be preferable to work with a heuristic method if the guarantee of optimality is not crucial. Heuristic procedures may be as simple as a single-pass ranking or they may be complicated by intricate rules for examining a large number of feasible solutions before stopping.

RSP contains a sophisticated, general heuristic procedure known as the Standard Evolutionary Engine (SEE). This procedure belongs to the class of procedures know as *genetic algorithms*, and its details are proprietary. The SEE is available as an option in RSP, and the limit of 200 variables is normally not a constraint in solving sequencing problems. Nevertheless, some previous work has indicated that the SEE performs particularly well on sequencing problems, so tests were included in the computational experiments to probe its performance.

Unlike optimization algorithms, which are intended to be implemented once to produce a solution, the SEE is most effectively implemented with a series of runs, normally with manual intervention to adapt the parameters of each run based on the outcome of the previous run. However, this mode of operation is unwieldy for experiments on a series of test problems. We implemented the SEE with the following automated steps, to approximate the results of manual intervention.

[1] For more information, see Frontline Systems' website, http://www.solver.com.

1. Initialize the sequence using a simple heuristic suitable for the problem type under study.
2. Run the SEE using its default parameters.
3. Re-run the SEE 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-initialize the sequence using a lexicographic 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.

In this implementation, the time limit was set to 10 seconds for each run. The SEE contains random components in its formation of solutions by mutation. Thus, no two runs from a given initial sequence can be guaranteed to produce the same result. Similarly, the six-step procedure used in this study may or may not produce a better solution than a fully-flexible manual implementation. However, our automated procedure likely underestimates the results attainable by manual implementation, which can involve many re-initializations and many tailored adjustments of the search parameters.

1.4. The Problems

Sequencing problems provide a good laboratory for exploring the capability of RSP. We ran experiments for some familiar sequencing problems:

- The single-machine tardiness problem
- The flow shop scheduling problem
- The traveling salesperson problem

These three problem types have several features in common. First, they are relatively easy to state and require relatively limited data, yet they are often challenging to solve. Second, they each have a history in the literature: they are well studied by several researchers. In fact, they have served as proving grounds for innovation in tackling difficult combinatorial problems. They also share some structural similarity: each problem calls for finding the best permutation. In addition, their formulations as mixed-integer programs all build on an imbedded assignment model.

The flow shop scheduling problem usually refers to a version in which the objective is to minimize the schedule length or *makespan*. However, a variation is to minimize the total completion time (equivalently, the mean flowtime), and we study that variation as well.

We explored two approaches for solving traveling salesperson problems with integer programming. In the first approach, called the *index method*, a model is built and solved once to produce the solution. The index method is inefficient on large problems, however, and a second approach, known as the *subtour elimination method*, calls for solving a series of integer programs, with constraints added at each iteration until the best sequence is found. The subtour elimination method handles larger problems than the index method, although it requires the solution of many integer programs en route to an optimal sequence.

2. The single-machine tardiness problem

The single-machine tardiness problem (SMTP) contains n jobs. Job j has a known processing time p_j and due date d_j. As a result of sequencing decisions, job j realizes a completion time C_j, and its tardiness is defined by $T_j = \max\{0, C_j - d_j\}$. The objective is to minimize the total tardiness, $\sum_{j=1}^{n} T_j$.

The SMTP has been studied extensively, and optimal solutions can be obtained by various methods. The state of the art appears to be represented in the work of Szwarc et al. (2001), who were able to solve problems containing several hundred jobs. In our experiments, we formulated the SMTP as a mixed-integer program using a formulation in which the key variables assign jobs to positions in sequence. Baker and Keller (2010) demonstrated the computational superiority of that formulation in comparison to several alternatives. The formulation of an n-job problem contains $n(n + 1)$ variables and $3n$ constraints. The educational version of the software accommodates problem sizes of up to 13 jobs.

For the purposes of testing solutions to the SMTP with the RSP software, we created a set of randomly-generated test problems characterized by a widely-used convention based on a *due-date range* and a *tardiness factor*. The due-date range (DDR) is the range of the job due dates, expressed as a proportion of the expected schedule length. The tardiness factor (TF) is the proportion of jobs that would be expected to finish tardy in a random sequence, as a function of the average due date. For each problem size n, we generated a test problem for standard combinations of due-date range (DDR = 0.2, 0.4, 0.6 and 0.8) and tardiness factor (TF = 0.2, 0.4, 0.6, 0.8 and 1.0), for a total of 20 test problems.

Table 1 summarizes our results for various problem sizes. The table shows the average, median, and maximum solution time, in seconds, along with the number of instances (out of 20) in which the SEE delivered an optimal solution. For the 13-job problems, all optimal solutions were obtained with the mixed-integer program within three seconds of cpu time.[2] Thus, the educational version of RSP handles problems with up to 13 jobs and find optimal solutions quickly.

Size (n)	13	15	20	25	30
Time (Avg.)	2.16	3.82	3.46	26.62	201
Time (Med.)	2.05	3.32	2.41	4.03	4.96
Time (Max.)	2.88	6.86	8.61	256.17	3600
ES optimal	20	20	20	20	19

Table 1. SMTP results.

For larger problems, solutions were achievable for problem sizes up to about 30 jobs. Even with n = 30, the median run time is about a five seconds, but considerable variability exists. One of the test problems could not be solved within the one-hour limit. Our results reinforce the empirical observation that the

[2] Risk Solver Platform reports three run times: parse time, setup time, and solve time. The figures in these tables represent the sum of the three times. As problem size varies, the parse time and setup time tend to be relatively fixed, and most of the variability reflects variation in solve time.

tardiness problem can be easy or difficult, depending primarily on the values of DDR and TF. The longest run times were observed for DDR at 0.6 or 0.8 and TF at 0.8.

In the SMTP we find that most instances can be solved readily, but we occasionally encounter an instance that takes orders of magnitude longer to solve. In the SMTP we can identify datasets that are likely to be easy or difficult by examining DDR and TF before attempting a solution. In other types of sequencing problems, we encounter similar variability in solution times, but we do not have as much insight into the factors that account for easy or difficult solutions. To explore that observation, we created a testbed of "easier" instances by holding TF at 0.2 and DDR at 0.2 and 0.4. Under those conditions, problems containing 40 jobs were solved in a matter of seconds, and problems containing 100 jobs were solved in an average of less than a minute.

In the test data containing 13, 20 and 25 jobs, the automated implementation of the SEE produced optimal solutions for every instance. In larger problems, it produced optimal solutions in the vast majority of instances, though not every time. (In the case of the "easier" 100-job problems, the SEE found optimal solutions in 60% of the instances.) In general, it appears that the SEE is well suited to obtaining very good results in the SMTP and often produces optimal solutions, with performance that deteriorates as the problem size becomes large.

As research on solution algorithms for the SMTP developed, Potts and Van Wassenhove (1985) wrote, "A recent algorithm allows total tardiness problems with up to 100 jobs to be solved within reasonable CPU times. Thus, at least for practical purposes, the total tardiness problem has been solved satisfactorily." Although the size of a problem that we might want to solve for practical purposes may be a subject for debate, the standard of 100 elements seems like a reasonable one. For that standard, we can achieve only a partial solution (i.e., for the "easier" problem instances) using a laptop and spreadsheet-based software. As we shall see, however, the story is more positive in other sequencing problems.

3. The flow shop problem with makespan objective

The flow shop problem contains n jobs and m machines. Each job must be processed by each of the machines in the same machine order, and the machines can process at most one job at a time. The k^{th} operation of job j has a known processing time p_{kj}. As a result of sequencing decisions, job j will have a completion time C_j. One objective is to minimize the maximum completion time, or *makespan*. We refer to this as the flow shop makespan problem (FSMP). Here, we actually address the permutation flow shop model, meaning that the job sequence is the same at all machines, so the schedule is determined by a single permutation of the jobs.[3]

The FSMP has been studied extensively, and optimal solutions can be obtained by various methods. The two-machine problem can be solved efficiently, but when $m \geq 3$, the problem is NP-Hard in general. The state of the art appears to be represented in the work of Ladhari and Haouari (2005),

[3] The three-machine FSMP has an optimal solution consistent with the permutation flow shop model, but that is not always the case when the problem contains four or more machines.

who were able to solve three-machine problems containing up to 2000 jobs. However, they also encountered problems as small as 100 jobs and 6 machines (100x6) that could not be solved within an hour of CPU time.

In our experiments, we formulated the FSMP as a mixed-integer program using a formulation in which the key variables assign jobs to positions in sequence. Tseng et al. (2004) and Stafford et al. (2005) demonstrated the computational superiority of that formulation in comparison to the main alternatives. The formulation of an n-job problem contains $n^2 + 2mn$ variables and $n(m+1)$ constraints. If we focus on the three-machine problem, the largest problem size that the educational version accommodates is 11 jobs. For "square" problems, in which $m = n$, the largest is 8x8.

For the purposes of testing solutions to the SMTP, we generated a set of test problems by sampling the mn processing times independently from a uniform distribution on the interval [1,99]. This approach appears to be a standard convention in the literature. For each problem size, 10 replications were carried out. Samples of size 10 may not give a complete picture of the variability of solution times, especially for larger problem sizes.[4] However, the results suggest where the "knee of the curve" sets in.

Size (n)	11	8	30	40	10	15	100
Size (m)	3	8	3	3	10	15	3
Time (Avg.)	1.71	1.97	2.05	2.07	3.90	579	5.45
Time (Med.)	1.70	1.98	1.99	1.97	2.91	362	4.34
Time (Max.)	1.81	2.05	2.41	3.07	14.39	1881	12.10
ES optimal	4	0	3	1	0	0	0
ES avg.	1.02	1.05	1.03	1.01	1.07	1.09	1.00
ES med.	1.01	1.05	1.02	1.01	1.07	1.09	1.00
ES max	1.10	1.10	1.08	1.03	1.13	1.12	1.01

Table 2. FSMP results.

Table 2 summarizes our results. Again, the problem sizes that can be accommodated by the educational version can be solved in less than three seconds, and relatively little variation in run times seems to occur. Larger three-machine problems are also solved quickly, even up to the 100x3 problems, where solutions can still be found in just a few seconds. Among square problems, 10x10 problems take only a few seconds to solve, but 15x15 problems begin to show the kind of variability noted earlier for the difficult SMTP instances. Although the median solution time for 15x15 problems was about six minutes, one instance took over half an hour to solve.

Another set of runs was aimed at the standard dataset developed by Taillard (1993). The flow shop problems in that dataset contain 20, 50, and 100 jobs, and 5, 10, and 20 machines. Although solutions have been found to all of the problems, even the best algorithms have been unable to solve all

[4] Reflecting on their computational experience with the flow shop model, Companys and Mateo (2007) remarked that even a sample of 1000 random instances may not be sufficient to capture the complete range of variability.

of the 10-job and 20-job problems with a one-hour limit on cpu time. That was also our experience with the integer programming model; however, each of the 30 five-machine problems (10 of which contained 100 jobs) was solved within an hour of cpu time.

The SEE is not as effective for the flow shop problem as it is for the SMTP, even though both give rise to models in which the decision variables correspond to a permutation of the integers 1 through *n*. The heuristic approach found only a few optimal solutions in the three-machine problems. On the other hand, its average suboptimality improved dramatically for larger problem sizes, to well under 1% for 100x3 instances. In the square problems, the SEE performance was not as good, with average suboptimality in the 5-10% range.

In the case of the FSMP, the spreadsheet-based approach is not quite as fast as the state-of-the-art algorithm, but it is reasonably competitive in terms of the problem sizes—particularly "practical" problem sizes—that it can accommodate.

4. The flow shop problem with total completion time objective

An alternative objective for the permutation flow shop is to minimize the sum of the completion times, $\sum_{j=1}^{n} C_j$. We refer to this as the flow shop completion-time problem (FSCP).

The FSCP has been solved by branch-and-bound techniques but has not been studied as extensively as the makespan problem. The two-machine problem is NP-Hard when the objective is minimizing the sum of completion times. The state of the art appears to be represented in the work of Akkan and Karabati (2004), who solved problems with 40 jobs and 2 machines without requiring more than an hour of cpu time. Researchers have reported results for more than two machines on very limited occasions. The most successful set of such experiments may be due to Chung et al. (2002), who solved problem sizes of 10x10 and 15x8. However, their runs were limited by a ceiling on the number of nodes in the search tree, and they were unable to solve all problems of size 20x4.

In our experiments, the FSCP was formulated as a mixed-integer program using the same basic structure as for the FSMP. The formulation of an *n*-job problem contains $n^2 + 2mn$ variables and $3n$ constraints. Thus, the largest two-machine problem size that the educational version accommodates is 12 jobs.

For the purposes of testing solutions to the SMTP, we generated a set of test problems by sampling the *mn* processing times independently from a uniform distribution on the interval [1,99]. Della Croce et al. (2002) observed that solution times are generally faster when the interval is restricted to [1,10], so we limited our experiments to the more challenging set. For each problem size, 10 replications were carried out.

Size (*n*)	12	11	8	20	25	30	32	20	10	15
Size (*m*)	2	3	8	2	2	2	2	4	10	15
Time (Avg.)	1.66	1.91	1.78	2.69	9.64	350	226	16.7	2.71	343
Time (Med.)	1.65	1.89	1.76	2.65	6.23	31	222	12.6	2.72	140

Time (Max.)	1.73	2.01	1.91	4.01	35.04	2709	724	57.9	3.27	1841
ES avg.	1.08	1.09	1.04	1.07	1.09	1.09	1.09	1.13	1.07	1.08
ES med.	1.10	1.08	1.04	1.07	1.07	1.08	1.10	1.13	1.06	1.08
ES max	1.15	1.17	1.06	1.13	1.18	1.13	1.12	1.17	1.12	1.13

Table 3. FSCP results.

Table 3 summarizes our results. Again, the largest problem sizes that the educational version accommodates (12x2, 11x3, and 8x8) are solved within two seconds. Larger two-machine problems are also solved quickly, but the computation times begin to grow substantially at problem sizes of 25 and 30. However, RSP can solve most instances of 35x2, which is close to the current state of the art. For problems with more than two machines, our results indicate that problem sizes of 20x4 and 15x15 can be solved with RSP. In other words, we were consistently able to solve two-machine problems about 80% as large as those at the state of the art, and our results on problem sizes of 20x4 and 15x15 go beyond the results reported in the research literature.

In terms of heuristic performance, the SEE achieved average suboptimalities in the range of about 5-10% for two-machine instances and produced no optima. In many cases, however, we noticed that the SEE solution could not improve on a simple sorting routine which sequenced the jobs in nondecreasing order of total processing time over all machines. This objective function apparently leads to a challenging problem for heuristic solution procedures.

5. The traveling salesperson problem

The traveling salesperson problem (TSP) contains n cities and is specified by an array of intercity distances, d_{jk}. The problem is to find a tour of the cities (visiting every city once) with minimal total distance.

The TSP problem is NP-Hard in general and has been studied very extensively.[5] Various sets of test problems have been published over the years, some with data from actual locations and others with data from a variety of applications in network design, manufacturing, and genetics. Test problems in the well-known library TSPLIB[6] contain sizes from 14 cities up to 85,900 cities. The largest of these problems has been solved by exploiting parallel computing methods and consuming the equivalent of several centuries of cpu time.

In our experiments, we formulated the TSP in two standard ways. First, we constructed a solution using the so-called *index method* of Miller, Tucker and Zemlin (1960) as strengthened by Desrochers and Laporte (1991). This approach produces a mixed-integer program containing $(n^2 + n - 1)$ variables and $(n - 1)(n - 2)$ constraints. With this formulation, the largest problem size that the educational version accommodates is 13 cities.

[5] A recent guide to the problem can be found at http://www.tsp.gatech.edu/index.html.
[6] Material on TSPLIB can be accessed at http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/.

A second set of experiments used the *subtour elimination approach*. This method first solves an assignment model with n^2 variables[7], corresponding to the given distance array and then checks whether the solution corresponds to a tour. If not, the solution must contain subtours, so constraints are added to prohibit the use of selected subtours, and the expanded model is solved. In principle, it is possible to add just one constraint and then re-solve the problem. However, it is generally more efficient to add several constraints when a subtour is found. This approach could still require an exponential number of cycles (solving one mixed-integer program at each cycle), but empirical evidence indicates that the number of constraints grows roughly in proportion to the problem size.

For the TSP we constructed test problems by randomly generating city locations on a two-dimensional grid and then computing the intercity distances. For each problem size, 10 replications were carried out.

Size (*n*)	13	20	25	30	35	40	44
Time (Avg.)	0.72	1.36	2.92	6.0	374	72	436
Time (Med.)	0.71	1.32	1.90	5.4	10.9	20	55
Time (Max.)	0.82	2.09	9.27	12.2	3597	245	3365
ES optimal	10	10	10	5	4	1	0
ES avg.	1.00	1.00	1.00	1.00	1.01	1.01	1.01
ES med.	1.00	1.00	1.00	1.00	1.00	1.01	1.01
ES max	1.00	1.00	1.00	1.00	1.02	1.04	1.03

Table 4. TSP results.

Table 4 summarizes results for the index method. The largest problem sizes that the educational version accommodates (13 cities) were solved in less than a second. Larger problems were also solved quickly, but the computation times began to grow substantially at problem sizes above 30. Problems containing 44 cities, which require almost 2000 variables can be solved with this model, although the time required may occasionally be quite long.

Meanwhile, it is interesting to observe that the SEE handled the TSP readily and often produced optimal solutions even for problem sizes up to 30 cities. As indicated in the table, the average suboptimality was consistently within 1%.

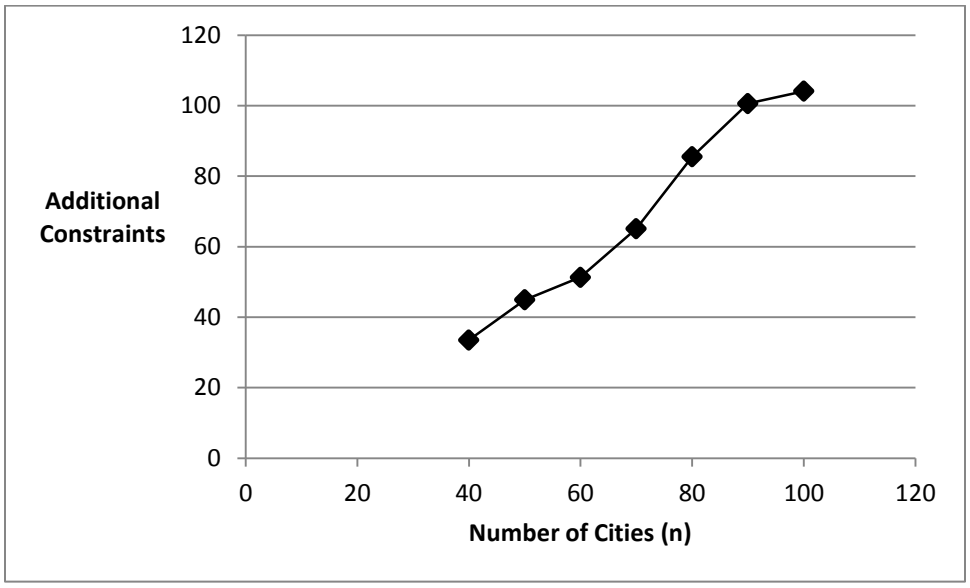
Size (*n*)	40	50	60	70	80	90	100
Cycles	4.4	7.1	5.8	7.9	10.0	20.0	16.6
Constraints	33.5	44.9	51.3	65.1	85.5	101	104

[7] Both formulations of the TSP in our experiments contain variables x_{jj}, which will be zero in the optimal solution. However, for ease in spreadsheet layout and model specification, these variables appear in the formulation and are counted in the problem size calculations.

Time (Avg)	12.34	19.89	15.52	22.01	122.7	96.21	89.89
ES optimal	7	4	3	0	0	0	0
ES avg.	1.003	1.01	1.01	1.02	1.02	1.04	1.05
ES med.	1.00	1.00	1.01	1.02	1.02	1.04	1.05
ES max	1.02	1.05	1.03	1.04	1.05	1.07	1.09

Table 5. TSP results with subtour elimination.

Table 5 summarizes results for the subtour elimination method. With this approach, RSP can solve problems with up to 100 cities without consuming inordinate amounts of cpu time. The table shows the average number of cycles, where each cycle involves appending a set of constraints and running the mixed-integer model once more, along with the average number of constraints ultimately added. Although cycle adds multiple constraints, the total number of constraints required to obtain a solution remains manageable, roughly at the same order of magnitude as the number of cities. (The chart in Figure 1 shows the relation between the average number of constraints in the model and the problem size. The chart reinforces the point that the number of constraints tends to grow approximately linearly with problem size.)



The chart suggests that the sample for 100 cities was a bit on the easy side, but the overall picture remains: the spreadsheet-based approach is viable up to (and probably beyond) 100 cities.

The SEE continued to perform quite well on these larger problems, although its results deteriorated as the problem size grew beyond 40 cities. At larger problem sizes, the 10-second time limit (maintained in Table 4) tended to be restrictive, so we increased the time limit to *n*/2 seconds for each *n*-city problem. With this adjustment, we found that the SEE performed quite well even for large

problems, as shown in Table 5. Its average suboptimality remained at about 1-2% for problems of up to 80 cities and rose just a bit higher on the larger problems (an effect that can be somewhat mitigated by using even larger time limits.)

6. Summary and Conclusions

We studied four different sequencing problems and their solutions using spreadsheet-based mixed-integer programs and RSP software. Interestingly, all four problems lead to models that seek the best permutation of *n* integers, and they can all be formulated as mixed-integer programs with an assignment kernel. Nevertheless, their computational features are somewhat different. We noted that the educational version of RSP accommodates 200 variables, which translates into problem sizes of up 13 tasks (in the SMTP and the TSP) and 11 tasks in the case of the three-machine flow shop. Problems of these sizes can be solved in at most a few seconds using the Gurobi Solver Engine, yet they are hardly "toy" problems: they represent useful models for the practitioner and challenging exercises for the student.

The larger version of the software gives rise to a different story. For the SMTP and the FSCP, optimal solutions begin to require prohibitive amounts of computation time at roughly 30-35 tasks. The index method for the TSP can extend only slightly beyond that size. On the other hand, for the three-machine FSMP, 100-job problems can be solved quickly by RSP software, as can a certain subset of SMTP instances. The subtour elimination approach to the TSP is not constrained by time limits in the usual fashion, and we were able to solve problems containing 100 cities. For our 100-city test problems, the subtour elimination approach requires the solution of about 15-20 mixed-integer programs, on average, each one requiring a few seconds on average. Although this approach could be executed manually, a fully-automated implementation of the subtour elimination method requires additional coding. In our experiments, the code was written in Visual Basic for Applications (VBA), which is a software language that many students learn in conjunction with Excel.

From a research perspective, the mixed-integer programming model is nearly competitive with state-of-the-art algorithms for the two flow shop problems, and it is capable of solving moderately sized versions of the SMTP and the TSP. These results suggest that as we encounter new types of sequencing problems, we may find that a reasonable computational approach may be to rely on generic mixed-integer programming models. Such a finding, in turn, reinforces the value of indcluding mixed-integer programming in the training of students who are likely to work on practical scheduling problems. They may not have to exploit the state of the art to solve such problems; rather, they may do quite well using a spreadsheet-based approach on their laptops.

References

Akkan, C. and S. Karabati (2004) The two-machine flowshop total completion time problem: Improved lower bounds and a branch-and-bound algorithm. *European Journal of Operational Research* 159, 420–429.

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

Chung C-S., J. Flynn and O. Kirca (2002) A branch and bound algorithm to minimize the total flow time for m-machine permutation flowshop problems. *Int. J. Production Economics* 79, 185-196.

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

Della Croce, F., M. Ghirardi and R. Tadei (2002) An improved branch-and-bound algorithm for the two machine total completion time flow shop problem. *European Journal of Operational Research* 139, 293-301.

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

Ladhari, T. and M. Haouari (2005) A computational study of the permutation flow shop problem based on a tight lower bound. *Computers & Operations Research* 32, 1831–1847.

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

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

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

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

Taillard,E. (1993) Benchmarks for basic scheduling problems. *European Journal of Operational Research* 64, 278-285.

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

Appendix. Formulations Used in the Experiments

SMTP

Notation

n = number of jobs

p_j = processing time for job j

d_j = due date for job j

Decision variables

x_{jk} = 1 if job j is assigned to sequence position k

t_k = tardiness of the job in sequence position k

Constraints

$$\sum_{u=1}^{n} x_{uk} = 1, \quad \text{for all positions } k,\ 1 \le k \le n$$

$$\sum_{u=1}^{n} x_{ju} = 1, \quad \text{for all jobs } j,\ 1 \le j \le n$$

$$\sum_{u=1}^{k}\sum_{j=1}^{n} p_j x_{ju} - \sum_{j=1}^{n} d_j x_{jk} \le t_k, \quad \text{for all positions } k,\ 1 \le k \le n$$

Objective

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

FSMP

Notation

n = number of jobs

m = number of machines

p_{ij} = processing time for job j on machine i

Decision variables

x_{jk} = 1 if job j is in sequence position k

I_{ik} = idle interval on machine i prior to the start of the job in sequence position k

H_{ik} = idle time for the job in sequence position k after finishing on machine i

Constraints

$I_{i,k+1} + \sum_{u=1}^{n} p_{iu} x_{u,k+1} + H_{i,k+1} - H_{ik} - \sum_{u=1}^{n} p_{i+1,u} x_{uk} - I_{i+1,k+1} = 0$, for all sequence positions k, $1 \le k \le n-1$ and all machines $1 \le i \le m-1$; and, for the first job,

$I_{i1} + \sum_{u=1}^{n} p_{iu} x_{u1} + H_{i1} - I_{i+1,1} = 0$, for all machines $1 \le i \le m-1$.

Objective: The makespan, less the sum of processing times on the last machine (which is constant), yields the sum of idle times on the last machine.

Minimize $\sum_{k=1}^{n} I_{mk}$

FSCP

Notation

n = number of jobs

m = number of machines

p_{ij} = processing time for job j on machine i

Decision variables

x_{jk} = 1 if job j is assigned to sequence position k

I_{ik} = idle interval on machine i prior to the start of the job in sequence position k

H_{ki} = idle time for the job in sequence position k after finishing on machine i

Constraints

$I_{i,k+1} + \sum_{u=1}^{n} p_{iu} x_{u,k+1} + H_{i,k+1} - H_{ik} - \sum_{u=1}^{n} p_{i+1,u} x_{uk} - I_{i+1,k+1} = 0$, for all sequence positions k, $1 \le k \le n-1$ and all machines $1 \le i \le m-1$; and, for the first job,

$I_{i1} + \sum_{u=1}^{n} p_{iu} x_{u1} + H_{i1} - I_{i+1,1} = 0$, for all machines $1 \le i \le m-1$.

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

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

Therefore the sum of completion times can be expressed in terms of the decision variables as follows:

$$\text{Minimize } \sum_{k=1}^{n} (n-k+1)[\sum_{j=1}^{n} p_{mj} x_{jk} + I_{mk}]$$

TSP (MTZ model)

Notation

d_{ij} = distance from city i to city j

Decision variables

x_{ij} = 1 if route from city i to city j is assigned to the tour

u_i = index for city i (effectively, the position in sequence), $2 \le i \le n$

Constraints

$$\sum_{i=1}^{n} x_{ij} = 1$$

$$\sum_{j=1}^{n} x_{ij} = 1$$

$u_i - u_j + (n-1)x_{ij} + (n-3)x_{ji} \le n-2$, for all pairs (i, j), $i, j \ge 2$ and $i \ne j$

$1 \le u_i \le n-1$

Objective

$$\text{Minimize } \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} x_{ij}$$

TSP (Subtour elimination model)

Notation

d_{ij} = distance from city i to city j

Decision variables

x_{ij} = 1 if the segment from city i to city j is assigned to the tour

Constraints

$$\sum_{i=1}^{n} x_{ij} = 1$$

$$\sum_{j=1}^{n} x_{ij} = 1$$

Given a solution that does not form a tour, it must contain at least one subtour S (interpreted as the set of segments (i,j) in the subtour). Let $|S|$ denote the size of set S. Then a subtour elimination constraint excludes the subtour (set) S:

$$\sum_{(i,j) \in S} x_{ij} \leq |S| - 1 \quad \text{for subtour } S$$

In other words, the number of segments assigned to this set of cities must remain less than the length of the subtour.

Objective

Minimize $\sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} x_{ij}$