Research Notes for Chapter 3*

In this chapter we meet for the first time an NP-hard problem that we can essentially solve to optimality by state of the art algorithms: the *T*-problem, for which we can now solve instances with up to 500 jobs. These solutions are based on a branch and bound approach utilizing clever tight bounds and dominance properties. The most influential source of such dominance properties is Emmons (1969), starting with Theorem 3.3 which is due to him. Another early source is Elmaghraby (1968), to whom Theorem 3.2 is due. We did not cover the whole gamut of Emmons' results, however. For instance, he develops a corollary that if EDD yields a sequence under which every job starts before its due date (which implies that $T_j \leq p_j$ for all j) then it minimizes tardiness. Compare the well-known property that when EDD yields only one tardy job, it is optimal. Emmons' corollary places no restriction on the number of tardy jobs, whereas the property places no restriction on the start time of the tardy job. We may also observe a similarity to the subsequent Wilkerson and Irwin result that we presented in RN2, in that the tardiness intervals cannot overlap and yet satisfy Emmons' corollary. Otherwise, the due date of the second tardy job within an interval would have to reside within the tardiness interval of the previous job, and then that second job would necessarily start *after* its due date. Thus, the Wilkerson-Irwin result is more general at least in the sense that it allows $T_j > p_j$. (Interested readers may wish to prove or disprove that the Wilkerson-Irwin result includes Emmons' result as a special case.)

Emmons' dominance conditions were subsequently generalized by Rinnooy Kan et al. (1975). Although it still took a lot of effort by many researchers to solve the problem, the *T*-problem is relatively easy in the computational sense because it is only NP-hard in the weak sense: Lawler (1977) presented a pseudopolynomial solution. Lawler (1982) demonstrated a polynomial approximate solution based on his earlier result. Recall from our Research Notes for Chapter 1 that this is often useful when a pseudopolynomial solution exists. All we need to do is round the processing times to relatively small numbers by using large enough time units such that we obtain a small size unary input. But we know that a pseudopolynomial solution is polynomial in the unary size of the input. So it can achieve an optimal solution for the rounded processing times. Given a rounded solution (i.e., a sequence that is optimal for the rounded value) we can calculate the value associated with it for the precise processing times and the result is an upper bound on the true optimal solution. If the original rounding is replaced by truncation, the optimal solution is a lower bound (and we can still obtain an upper bound as before). A good approximate solution is achieved if the difference between the lower bound and the upper bound is sufficiently small. Technically, if we can guarantee that the gap between the bounds is smaller than ε (> 0) times the lower bound, and if we can do this for any ε, then we can also say that the solution is an ε-approximation of the optimal solution. Lawler (1982) exploited the existence of the pseudopolynomial solution he had developed for the *T*-problem in Lawler (1977) to

---

generate tight bounds for the problem in polynomial time, although the polynomial in question was quite time consuming.

In contrast to the *T*-problem, however, the T_w-problem is NP-hard in the strict sense. Recall that some of the dominance conditions can be extended to the weighted case, especially when parameters are agreeable (that is, jobs with lower processing times and lower due dates have higher weights). Potts and Van Wassenhove (1985) gave a B&B solution that also makes use of DP within branches (to prevent local violations of precedence relationships). They reported reliable solutions with up to 40 jobs, but 50-job problems sometimes caused severe computational difficulty. Although their work is over twenty years old, we should not expect today's computers to handle much larger problems with that algorithm, either. (A likely order of magnitude of increase is about 10 jobs.) The dynamic programming approach that we discussed with respect to the *T*-problem can also handle weighted tardiness, but only smaller problems can be addressed because in the weighted case it is more difficult to generate precedence relationships. The limiting factor in dynamic programming, in spite of the exponential time requirement, is storage. As discussed in Section 3.4, however, dominance conditions can reduce the storage requirement, and that effect can be dramatic. In one reported case, Schrage and Baker (1978) demonstrated a storage reduction from almost 1.7 million to less than 20,000; i.e., an 85-fold reduction. Rachamadugu (1987) gave a generalized version of MDD for two consecutive jobs. He further developed his result to demonstrate that if the SWPT sequence results in all jobs tardy then it is optimal. Kanet (2007) provides several additional results along similar lines, specifically encompassing constraints that apply to non-adjacent jobs. These results generalize and enhance earlier ones by Emmons and by Rinnooy Kan et al. Kanet and Li (2004) reported simulation results that use a slightly simpler generalization of MDD. We discuss some of these contributions in Chapter 4, in the context of search heuristics. For a fairly extensive list of papers on both versions of the tardiness problem published before 1998, we refer to Sen et al. (2003). Earlier surveys include Koulamas (1994) and Abdul-Razaq et al. (1990), who focused on the unweighted and weighted versions, respectively. However, a breakthrough in solving the T_w-problem is represented by the recent work of Pan and Shi (2007) and Bigras et al. (2008). The core idea in these two papers is to represent the problem by a time-indexed integer programming model (see Appendix C), and then to compute bounds by linear programming models, which are inherently easier to solve. Both papers mention the usefulness of incorporating precedence conditions during the branching stage. With tight bounds to guide the branching, good results can be obtained. In fact, Pan and Shi report solving all 100-job open research problems, most within 4 hours and one within 9 hours on a 2.8 GHz PC.

Now that the test problems that resisted solution for over 20 years have all been solved, future research on the same problems will have to demonstrate shorter computation time, and it may be required to generate a set of larger benchmark problems as well. One question that such future research might test is whether the precedence conditions given by Kanet, which seem to be stronger than previous ones, can improve the state-of-the-art results. As is always the case in such tests, the answer depends not only on the strength of the conditions but also the price they may entail in computation time.

The *T*-problem is not the only problem that is NP-hard in the ordinary sense and yields to effective solution. The U_w-problem is another example, and it too can be solved efficiently in practice. The proof of its complexity status stems from Lawler and Moore (1969), even though their paper precedes the definition of the NP-complete

class. They showed a restriction to the knapsack problem by setting a common due date (so the problem reduces to selecting a set of jobs with different volumes and different weights to maximize the total weight in a given total volume; i.e., the knapsack problem). The knapsack problem, however, was identified as a member of the NP-complete class right from the start. Thus, indirectly, Lawler and Moore "proved" membership in the NP-complete class. They also gave a pseudopolynomial solution, demonstrating—again "in advance"—that it can't be NP-complete in the unary sense. Potts and Van Wassenhove (1988) provided an effective solution procedure that can handle most instances with up to 1000 jobs.

In practice, it is not always possible to wait until an optimization algorithm converges to the optimal solution, but if we maintain both upper and lower bounds during the process, we may choose to stop when they are close enough to each other. We then have the upper bound as our explicit solution and by comparing it to the lower bound we can provide a *certificate* of how close we are to optimality. This makes it possible to use the exact branch and bound solution but to stop when processing time exceeds a limit or when the gap between the bounds is small enough or when the gap is not closing fast enough—that is, when there is indication that insisting on full optimality may require excessive time relative to the potential benefit associated with it. In fact, most branch and bound applications include such stopping criteria. For this reason, branch and bound is still potentially useful as a heuristic even when running it to completion in every case may be prohibitive. NP-hardness only shows that the worst case computation time is excessive, so it may be possible to obtain optimal solutions often and good certificates otherwise. But if we require that a certificate with a pre-specified ε, then there is no guarantee that we can achieve that in polynomial time. Good heuristic solutions that provide tight upper bounds from the start improve the certificates. Tight lower bounds are equally effective. (Szwarc et al. 2001, whose algorithm is the state-of-the art solution of the *T*-problem, discuss an exception. But their algorithm involved memorizing formerly solved subproblems, and when bounds were too tight the quality of the stored problems deteriorated. So this is an exception that proves the rule.) Perhaps the most important single benefit of a tight gap between bounds is that branching is reduced: when the gap is small, many discrete variables can be fixed when we observe that reversing them leads to a lower bound above the current upper bound.

The high value of a good lower bound motivates an important approach exemplified by Fisher (1976), Potts and Van Wassenhove (1985) and Potts and Van Wassenhove (1988)—that is, *Lagrangian relaxation*. In Lagrangian relaxation of a minimization problem subject to constraints, we construct a related problem, often called the *dual*, or the Lagrangian, whose maximum solution provides a lower bound for the optimum of the original problem, called the *primal*. We may then use that bound in B&B to improve the search. In more detail, the idea is a direct generalization of the classic Lagrange multiplier method to combinatorial problems. In the classic Lagrange multiplier approach all constraints are given by equalities and we express each of them by an equivalent one that is constrained to zero by subtracting the right-hand-side from both sides. We then multiply the constraint by an appropriately selected multiplier and incorporate the product in the objective function. In effect, we add a weighted proportion of the constraint's violation to the main objective function. The weighing is associated with the magnitude of the multiplier. When the multiplier is selected optimally, the constraint has zero violation so the objective function reaches its optimum level without violating constraints. The multiplier then reflects the marginal value of each unit of constraint. For our purpose, we need to generalize Lagrange

multipliers to apply to inequality constraints and we need to show how to use them in a combinatorial optimization setting. If some constraints are given as inequalities, a nonzero multiplier is only necessary for those constraints that would be violated otherwise. If we express all inequalities as ≤ 0, then for the purpose of maximizing the dual objective, multipliers become positive for binding constraints. As a result, in a feasible solution the product of the multiplier and the associated violation will always be zero, either because the constraint is not binding and therefore the multiplier is zero or because the violation is zero. Shadow prices in linear programming (LP) are special cases of the multipliers. In general, a shadow price in LP is either zero or it is associated with a binding constraint. Typical scheduling problems, however, are more complex than LP because they involve discrete decision variables. To address such problems, we first formulate the problem as a mathematical model with appropriate objective function and constraints (see Appendix C for some examples). Next, we incorporate a subset of constraint violations into the objective function, with appropriate Lagrange multipliers. Variables that are constrained to be discrete (e.g., with integrality constraints) are typically just relaxed so the relevant variable is allowed to be continuous. For instance, a constraint of the type $x \in \{0, 1\}$ is replaced by $0 \leq x \leq 1$. Now the problem is solved with Lagrange multipliers taking care of the constraints incorporated into the objective function. However, due to the relaxed discrete constraints, the result is likely to be infeasible and thus provide only a bound. To find the tightest possible lower bound, we can adjust the multipliers; that is, we solve for the best multipliers. Those best multiplier values can typically be used to guide the branching that is next necessary to resolve violated discrete constraints. In effect, this approach combines continuous optimization (when solving for bounds) with discrete optimization (branching to resolve discrete constraint violations).

The breakthrough approaches we mentioned before are similar in the sense that they start with a mathematical model and use bounds obtained by solving linear programs to guide a B&B procedure. Such methods have been developed for generic problems, and those papers stress that they are applicable in scheduling too. A related approach is the *branch and cut* method. *Cutting plane* methods were historically developed to solve integer programs by the simplex method for linear programming. Essentially, we examine the current relaxed optimal solution and then add linear constraints designed to exclude (cut) the current non-discrete solution without excluding any discrete solutions. Experience revealed, however, that such cutting constraints are very useful up to a point, but they exhibit decreasing benefit. In such cases, it becomes expeditious to move to a different starting point by branching on persistent discrete variables (with a branch for each discrete value). This combination is called branch and cut. For example, the current benchmark solution of the TSP problem, Concorde—which we discuss in more detail in RN8—is a branch and cut application (see Applegate et al., URL). In our own experience, Concorde typically solves 1000-city randomly-generated instances within reasonable time. As any other branch and cut or branch and bound algorithm, however, Concorde tends to take longer when many nearly-optimal solutions exist, and thus it becomes difficult to fathom branches. Typical 100-city examples that we constructed took less than 20 seconds in the majority of cases, but in a particular set of experiments we generated a large number of nearly identical solutions one of which was optimal. In one such case, Concorde took 14 hours to solve a 100-city case. However, in a practical sense, when many near-optimal solutions exist, the problem is actually *easy*. Typically, in such a case, we quickly obtain upper bound solutions that are very close to the lower bound, and we thus easily obtain excellent certificates. In our own experiments we did not study larger

instances, but solutions of instances with dozens of thousands of cities have been reported, although such very large instances invariably utilize parallel processors and the computations can span many weeks. Finally, modern branch and bound methods, and their relevant variations, are important to a wide variety of combinatorial problems, not only for sequencing and scheduling. Current research tends to focus either on specific problems or on parallel processing. We can safely say that parallel processing is an important part of large scale discrete optimization (Ralphs et al., 2003), as we have already mentioned with respect to Concorde.

Our coverage of dynamic programming cannot be considered thorough, but readers can find much more on this subject in any academic library, starting with Bellman (1957). To appreciate Bellman's historical role in developing dynamic programming, see Dreyfus (2002). Dreyfus—who had worked extensively with Bellman—has also written two texts on the subject, and we cite the more recent one below. In this connection, when discussing the optimality principle of dynamic programming, we stated that it is satisfied by additive functions. Equivalently, it is also satisfied by functions that can be represented as additive functions (e.g., we can represent products of positive numbers by using logarithms to make them additive). But we do not wish to imply that dynamic programming cannot be implemented in wider contexts. For example, it can also be applied to solve max or min problems; e.g., when we append a job to a subset, the new maximum is max{former maximum, job value}. We chose not to mention within the chapter that dynamic programming can be used to solve max or min problems because it does not extend directly to corresponding stochastic cases, and it is not necessary for the objective functions that we study in the deterministic case. However, it is important to note that one of the objectives of Bellman when developing dynamic programming was precisely the ability to address stochastic problems. Bellman—a brilliant mathematician by all accounts—became interested in Operations Research during a period when most mathematicians did not think highly of applied mathematics. Our quote at the top of the research notes of Chapter 2 is excerpted from Dreyfus (2002), which in turn excerpted it from Bellman's autobiography. We conclude with another quote from the same source, speaking about the difficulty of solving realistic problems with analytic models:

> In order to make any progress, it is necessary to think of approximate techniques, and above all, of numerical algorithms.

Taking heed of this advice, our next chapter focuses on heuristics designed to address problems that are too difficult otherwise. Later, in Chapter 6, we introduce a numerical approximate approach to stochastic problems by simulation.

References

Abdul-Razaq, T.S., C.N. Potts and L.N. Van Wassenhove (1990) A Survey of Algorithms for the Single Machine Total Weighted Tardiness Problem, *Discrete Applied Mathematics* 26, 235–253.

Applegate, D., R. Bixby, V. Chvatal and W. Cook (URL accessed June 2008), http://www.tsp.gatech.edu/concorde

Bellman, R.E. (1957) *Dynamic Programming*, Princeton, NJ.

Bigras, L-P., M. Gamache, and G. Savard (2008) "Time-Indexed Formulations and the Total Weighted Tardiness Problem." *INFORMS Journal on Computing* 20, 133-142.

Della Croce, R., R. Tadei, P. Baracco and A. Grosso (1998) "A New Decomposition Approach for the Single Machine Total Tardiness Scheduling Problem," *Journal of the Operational Research Society* 49, 1101-1106.

Dreyfus, S.E. (2002) "Richard Bellman on the birth of dynamic programming," *Operations Research* 50, 48-51.

Dreyfus, S.E. and A.M. Law (1977) *The Art and Theory of Dynamic Programming*, Academic Press, New York.

Elmaghraby, S. E. (1968) "The One-Machine Sequencing Problem with Delay Costs," *Journal of Industrial Engineering* 19, 105–108.

Fisher, M. (1976) "A Dual Algorithm for the One-Machine Scheduling Problem," *Mathematical Programming* 11, 229-251.

Kanet, J.J. (2007) "New Precedence Theorems for One-Machine Weighted Tardiness," *Mathematics of Operations Research* 32, 579-588.

Kanet, J.J. and X. Li (2004) "A Weighted Modified Due Date Rule for Sequencing to Minimize Weighted Tardiness," *Journal of Scheduling* 7, 261-276.

Koulamas, C. (1994) "The Total Tardiness Problem: Review and Extensions," *Operations Research* 42, 1025–1041.

Koulamas, C. (1997) "Polynomially Solvable Total Tardiness Problems: Review and Extensions," *Omega* 25, 235-239.

Lawler, E.L. and J.M. Moore (1969) "A Functional Equation and its Applications to Resource Allocation and Sequencing Problems," *Management Science* 16, 77-84.

Lawler, E.L. (1977) "A 'Pseudopolynomial' Algorithm for Sequencing Jobs to Minimize Total Tardiness," *Annals of Discrete Mathematics* 1, 331-342.

Lawler, E.L. (1982) "A Fully Polynomial Approximation Scheme for the Total Tardiness Problem," *Operations Research Letters* 1, 207-208.

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.

Potts, C.N. and L.N. Van Wassenhove (1985) "A Branch and Bound Algorithm for the Total Weighted Tardiness Problem," *Operations Research* 33, 363-377.

Potts, C.N. and L.N. Van Wassenhove (1988) "Algorithms for Scheduling a Single Machine to Minimize the Weighted Number of Late Jobs," *Management Science* 34, 843-858.

Rachamadugu, R.M.V. (1987), "A Note on the Weighted Tardiness Problem," *Operations Research* 35, 450-452.

Ralphs, T.K., L. Ladányi and M.J. Saltzman (2003) "Parallel Branch, Cut, and Price for Large-Scale Discrete Optimization," *Mathematical Programming* 98, 253-280.

Rinnooy Kan, A.H.G., B.J. Lageweg and J.K. Lenstra (1975) "Minimizing Total Costs in One-Machine Scheduling," *Operations Research* 23, 908-927.

Schrage, L. and K.R. Baker (1978) "Dynamic Programming Solution of Sequencing Problems with Precedence Constraints," *Operations Research* 26, 444-449.

Sen, T., J.M. Sulek and P. Dileepan (2003) "Static Scheduling Research to Minimize Weighted and Unweighted Tardiness: A State-of-the-Art Survey," *International Journal of Production Economics* 83, 1-12.

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

Van Wassenhove, L.N. and L. Gelders (1978) "Four Solution Techniques for a General One-Machine Scheduling Problem," *European Journal of Operational Research* 2, 281-90.

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