Research Notes for Chapter 4*

After discussing sources for the results in Chapter 4, we present four techniques that we did not include in the chapter. Next, we present more details about dominance relationships for the T_w-problem and we discuss the effects of adding precedence constraints. Our objective is to encourage further research on those issues. (In Research Notes for Chapter 6 we discuss the stochastic version of the problem, with the same objective.) We also discuss asymptotic optimality and the related concept of asymptotic convergence. An important requirement in testing heuristics involves the design of appropriate test sets, so we explore that issue, too. We also provide the optimal values of the 12 test problems that we use in the chapter.

Some sources and credits

Dispatching: Beyond the results we reported in the chapter, the WMDD heuristic has been tested with simulation experiments, in both static and dynamic scenarios, and it has performed very well relative to other heuristics (Kanet and Li, 2004). The test based on an exact generalization of MDD was proposed by Rachamadugu (1987). We edited Rachamadugu's test slightly to match the format of SWPT. On the one hand, this test could be replaced by a straightforward API comparison. On the other hand, it is valuable as a basis for more powerful results, such as the property that if all jobs are tardy under SWPT then SWPT is optimal. We discuss other examples later.

Simulated annealing: The seminal SA work is Kirkpatrick et al. (1983). They discuss the basic insight that underlies SA, namely, the similarity between statistical mechanics and combinatorial optimization. The specific example that provides the most useful analogy is the annealing of solids (hence the name, simulated annealing). If the cooling regime is too fast (as in quenching), material will freeze into an inefficient state (with defects, or high energy). The analogy in optimization is settling for a low-quality local optimum. By contrast, a slow, controlled cooling regime leads to a low-energy state without defects, analogous to a good solution. The authors then describe the application of the main SA idea to the design of computer chips (which is a very complex combinatorial optimization problem) and to the traveling salesperson problem. A more detailed treatise of SA for combinatorial optimization is provided by Van Laarhoven and Aarts (1987).

Tabu search: The seminal work in this area is Glover (1989) and (1990a), although by the time these papers were published, quite a few applications of TS had already been completed based on earlier working papers. A very accessible introduction to TS is given by Glover (1990b). This is a useful source for readers who want to gain a more thorough introduction to the subject. It covers important issues related to TS that we did not cover, such as aspiration criteria that make it possible to override a tabu and the use of long-term memory. In addition, there is a more thorough discussion of various types of tabu moves and the ideal size of a tabu list. TS is not specific to scheduling, but Glover (1990b) cites at least one scheduling example.

Genetic Algorithms: Like TS and SA, GA is not specific to scheduling. Work in this field flourished after the publication of Holland (1975), which might therefore be considered an early and important milestone in the development of GA. Nonetheless, the main GA ideas were explored by various research groups starting in the 1960s. We refer the readers to Wikipedia for the history of GA. Important scheduling applications of GA seem to have appeared starting in the mid 1980s. Specifically, in 1985, papers addressing the job shop and the traveling salesperson problem, as well as a paper on bin-packing (which is closely related to parallel machine scheduling) were presented at the 1st International Conference on Genetic Algorithms, at Carnegie-Mellon University (Grefenstette, 1985). We are not aware of earlier publications of GA in scheduling, but there is such a vast literature in this field that it is difficult to say with certainty that the 1985 proceedings are indeed first.

Beam search

If we were to implement a branch and bound algorithm as in Chapter 3 but without using any bounds, the tree of partial schedules would eventually generate all $n!$ complete sequences. Proceeding from the initial node, level k would contain all partial sequences of size k. Thus, for an n-job problem, level 1 would contain n partial sequences, level 2 would contain $n(n-1)$, level 3 would contain $n(n-1)(n-2)$, and so on. Thus, the tree would get wider at each level, becoming quite wide by the time it reaches the final level.

Beam search is a means of searching the tree in heuristic fashion, without allowing the width of the tree to become excessive. Beam search limits the number of partial sequences considered at each level to a given number b, called the *beam width*. Usually, the parameter b is a relatively small number, so that partial sequences may have to be discarded in order to keep within the beam width. A key requirement is a means for selecting partial sequences to retain at any level. Typically, we compute a lower bound for each partial sequence and keep only those with the b best bounds. In a sense, these are the most promising partial sequences, and it makes sense to "bet" that they will lead to the best complete solution. However, there are no guarantees that the best bounds encountered early in the search eventually lead to optimal solutions, so the algorithm is ultimately a heuristic procedure. It is also possible to generate upper bounds for each branch, using construction or dispatching heuristics. Such upper bounds might also be used for selection, but there is no guarantee that they would lead to optimal solutions,

either. Morton and Pentico (1993) discuss such options in some detail, but we are not aware of explicit research comparing them in practice.

By limiting the width of the tree to b, the computational complexity of the search becomes polynomial—$O(bn^3)$ or higher, depending on the exact design. By discarding some partial sequences at each level, we may the search away from the optimum. Thus, the quality of the solution depends on whether we can find a way to select the b branches so that they are likely to include the optimal or a near-optimal solution.

ALGORITHM RN4.1
Beam Search

Step 1. Select the beam width b. Let $k = 0$, and initialize the search with the empty partial schedule.

Step 2. Increase k by 1. Let b' denote the number of partial schedules available from the previous level (for $k = 1$, $b' = 1$). Generate and evaluate the $b'(n - k + 1)$ possible new partial schedules. If these are complete schedules, identify the best one and stop.

Step 3. Select the best b partial schedules to pursue further and return to Step 2.

Example RN4.1. Consider a problem containing $n = 5$ jobs with known processing times and due dates.

Job j	1	2	3	4	5
p_j	2	3	1	6	4
d_j	11	7	5	10	6

Consider the T-problem for the data in Example RN4.1. For tactical reasons, we adopt a simple version of the algorithm, in which we evaluate a partial sequence simply by calculating its total tardiness. We construct the sequence from the last position forward, and we use $b = 2$.

At level 1, for the last sequence position, there are five job candidates. They generate tardiness values as shown in parentheses when they complete at time 16:

XXXX1 (5) XXXX2 (9) XXXX3 (11) XXXX4 (6) XXXX5 (10)

The two best partial sequences are XXXX1 and XXXX4, so we keep those and discard the others.

At level 2, there are four job candidates for the next to last position, for both of the two partial sequences. Again, total tardiness values are shown in parentheses.

XXX21 (12)	XXX31 (14)	XXX41 (9)	XXX51 (13)
XXX14 (6)	XXX24 (9)	XXX34 (11)	XXX54 (10)

The two best partial sequences are XXX14 and either XXX41 or XXX24. However, there is no advantage to continuing with XXX41 because it contains the same scheduled jobs as XXX14 but with higher total tardiness. So we select XXX14 and XXX24.

At level 3, the candidates are as follows.

XX214 (7)	XX314 (9)	XX514 (8)
XX124 (9)	XX324 (11)	XX524 (10)

The two best partial sequences are XX214 and XX514. At level 4, there are two additional partial sequences for each of these:

X3214 (7)	X5214 (7)	X2514 (8)	X3514 (8)

Retaining the two best partial sequences and completing them, we obtain the two solutions 5-3-2-1-4 and 3-5-2-1-4, both of which achieve $T = 7$ (which is optimal here).

For illustration, we solved the 20-job test problems using different choices of beam width and a lower bound comparable to equation (3.5). The results are reported in Table RN4.1, which also reproduces the results for three basic construction procedures discussed in the chapter.

Table RN4.1

Algorithm	*Optimizing Frequency*	*Average Ratio*	*Maximum Ratio*
Greedy	0 of 12	1.22	1.39
Insertion	0 of 12	1.20	1.44
WMDD	4 of 12	1.02	1.10
Beam search ($b = 2$)	0 of 12	1.21	1.38
Beam search ($b = 5$)	0 of 12	1.19	1.34
Beam search ($b = 10$)	0 of 12	1.18	1.34

At least for this test data, the results indicate that the performance of beam search is roughly comparable to that of the simplest heuristic methods, and the choice of beam width does not make much of a difference.

As with other heuristic procedures, there is no guarantee that beam search will find an optimal solution. Two critical tactics are the beam width (a wider beam is safer but also more time consuming) and the mechanism for evaluating partial schedules. In Example RN4.1 and in the computational comparison, we used relatively simple evaluation mechanism. A more ambitious version of the algorithm would compute more powerful lower bounds with the hope of improved performance. In addition, because WMDD performed much better in our experiments, we could use WMDD within the

beam search to produce tight upper bounds. Then, using these upper bounds to evaluate the candidates, the results would likely improve relative to Table RN4.1. (Incidentally, in Example RN4.1, where WMDD reduces to MDD, it, too, achieves the optimal value of 7.)

Dynasearch

Consider the pair interchange (PI) neighborhood in a single machine environment. There are exactly $n(n-1)/2$ neighbors for each permutation sequence; that is, $O(n^2)$. A similar observation applies for the all insertion (AI) neighborhood. Dynasearch belongs to a small set of known heuristics that attempt to cover much larger neighborhoods, but still require only a polynomial effort per step to do so (Congram et al., 2002). In the case of dynasearch, the neighborhood size is $(2^{n-1}-1)$; that is, $O(2^n)$. Nonetheless, the search effort per step is $O(n^3)$. That is, it takes $O(n^3)$ to identify the best immediate neighbor out of $O(2^n)$ candidates. By contrast, in PI we enumerate all the immediate neighbors that we can check. As a result, more permutations are directly accessible and the potential for finding a good local minimum is higher. In addition to searching a large neighborhood, the authors attributed part of the reported success to using *iterated* searches. In iterated search, instead of generating multiple seeds randomly, new seeds are generated from the best known current solutions. To this end, *kicks* are used to perturb a solution sufficiently to allow searching a new neighborhood without returning to the former sequence. Conceptually, such kicks are similar to mutations in GA. In the particular case of the T_w-problem reported by Congram et al., use is also made of problem specific speedups (shortcuts) that may not apply in other applications, but the reported results are competitive even without those shortcuts. We now briefly describe the *basic* neighborhood, without multiple seeds and without shortcuts, and how it is searched efficiently from a given initial sequence.

The basic dynasearch neighborhood is embedded in a regular PI neighborhood. Unlike simulated annealing or tabu search, dynasearch utilizes the steepest descent in each step, and stops when no improving moves are available. However, at each single step, dynasearch considers not only single PI steps but also combinations (series) of PI steps, provided that if jobs [i] and [k] are interchanged and $i < k$, there is no other interchange involving job [j] for any $i < j < k$. For instance, if we start with the sequence 1-2-3-4, we can move to 2-1-4-3 in one step (by performing PIs on 1&2 and 3&4), but we cannot interchange 1&3 and 2&4 in one step, because 2 is between 1&3 and because 3 is between 2&4. In other words, to reach 3-4-1-2 requires two steps. Similarly, 4-3-2-1 is not a direct neighbor because 1&4 are interchanged so 2&3 should retain their original order. When all interchanges are compliant, we say that they are *independent*. A single step can involve any number of independent exchanges. As a result, the single best PI may have to be given up in favor of two or more individually less attractive interchanges that are better when combined. For example, if the single best PI involves a job that is between two jobs that are part of the best combination, the single best PI will not be made in that step.

To identify the best combination of independent interchanges in polynomial time, an efficient dynamic programming (DP) approach is invoked. The procedure has n stages, corresponding to jobs in the seed order: in stage k we consider the subset of jobs {[1], . . . , [k]}. For convenience, assume that the jobs are indexed by their current sequence, that

is, $j = [j]$ for all j. For subset {1}, we just record the tardiness contribution of job 1 when it is actually scheduled first. Next, we consider the subset {1, 2}, and compare the total tardiness contribution with and without interchanging the two jobs. We record the decision and the minimal total contribution of this subset. When we add job 3, we have three options.

- Leave the subset {1, 2} as before (with or without an interchange) and just append job 3 at the end. In such case, the total tardiness contribution is obtained by adding T_3 to the minimal total tardiness of subset {1, 2}.
- Interchange jobs 1 and 3 thus obtaining the partial sequence 3-2-1. In such case we calculate the total tardiness directly.
- Interchange jobs 2 and 3, thus obtaining the partial sequence 1-3-2. The total tardiness of this option is the previously recorded total tardiness of the preceding subset, {1} plus the total of 2 and 3 (in their interchanged order and starting after 1 completes).

Given the three feasible options, we select the best for subset {1, 2, 3}, and we record the decision and the total tardiness of the subset. At this stage we have information for three subsets, {1}, {1, 2}, and {1, 2, 3}. We note that sequences 3-1-2 and 2-3-1 cannot be considered in stage 3. As for the two sequences 1-2-3 and 2-1-3, we automatically consider the better one by appending 3 to the end. Generally, by the time we consider job k, we have the optimal values and decisions for all subsets {1}, {1, 2}, . , {1, 2, . , $j-1$}, {1, 2, . , $j-1$, j}, … , {1, 2, … , $j-1$, j, … , $k-1$}. We now consider interchanging job k with each one of the preceding jobs (which requires $O(n)$ steps). When we consider the interchange with job j (for $j > 1$), we use our recorded information to calculate the contribution of the subset {1, 2, … , $j-1$} and add the total tardiness contribution of the subset {j, … , k} in the specific sequence k-(j + 1)- … -(k − 1)-j, starting at the completion time of subset {1, 2, … , $j-1$}. After performing this calculation for job n, we have the information we need to select the best combination of PIs, thus completing a step. We then repeat the process for the new sequence. The search stemming from the original seed stops when no single PI exists that achieves an improvement over the former series of interchanges. (Thus, the final sequence is locally optimal not only in the dynasearch neighborhood but also in the PI neighborhood.)

Several sources, including Congram et al. (2002) and Congram (2000), also discuss other dynasearch neighborhoods. In particular, an enhanced dynasearch neighborhood allows insertions in addition to PIs. For instance, the sequence 3-1-2 cannot be considered at stage 3 in the basic dynasearch neighborhood, but it becomes eligible in the enhanced neighborhood by inserting job 3 in the first position. Similarly, the sequence 2-3-1 is eligible by inserting job 1 after job 3. More generally, in this enhanced dynasearch neighborhood, at stage k, in addition to the basic options, we now also allow job [k] to be inserted in the jth sequence position, provided it is followed by jobs [j], [j + 1], [j + 2], … , [k − 1] in that order, with no other job between any two of them. Similarly, any job j can be inserted right after job [k] provided jobs [j + 1], [j + 2], … , [k − 1], [k], [j] are then scheduled in the particular order given here, with no other jobs between any two of them. Grosso et al. (2004) test this enhanced neighborhood. They achieve even better results for large problems, but the approach is slightly slower

for smaller instances. Nonetheless, the time required to achieve an excellent level of performance is very short for both neighborhoods, so the speed issue is somewhat academic. For instance, Grosso et al. report matching the best known results [by then] for 125 different 100-job instances within 3 seconds per instance, on average (on a 800 MHz PC). They also reported that very extensive runs failed to improve on those formerly best known results. Fifty-job instances require about half a second. Recall from our Research Notes for Chapter 3 that it took up to nine hours of CPU time on a 3.5-fold faster 2.8GHz PC to obtain proven optimal solutions for these same problems by the state-of-the-art method (Pan and Shi, 2007). Furthermore, Pan and Shi report that those formerly known best results turned out to be optimal. That is, dynasearch is not only a highly efficient heuristic but also an extremely effective one.

We conclude our discussion of the dynasearch results with two notes. First, it is also possible to use insertion *instead* of PI rather than as an enhancement. However, Grosso et al. report that this tactic provided no advantage in their experiments. Second, they also report that in their tests, a multi-start approach worked better than the iterated search (with kicks) that Congram et al. recommended. The question why the two experiments led to such different conclusions with respect to this choice may justify more research attention.

In both the basic dynasearch neighborhood and the enhanced one, the restriction to independent steps is necessary to allow finding the best combination for any subset in polynomial time. In terms of the basic case, we would not be able to simplify the calculation involving the interchange of j and k by only considering the sequence k-(j + 1)- … -(k − 1)-j without independence. Another key requirement is that the effect of independent PIs on the objective function should be additive or have some other form amenable to DP recursive analysis (but such forms seem irrelevant here). For instance, consider the problem of scheduling n jobs on parallel machines, without preemption, to minimize makespan (see Chapter 9). It is possible to specify an optimal solution by *list scheduling*, which essentially employs a permutation to load the machines. The next job in the list is loaded as early as possible on the first free machine. Hence the problem boils down to finding the best permutation, and we might be interested in employing dynasearch to do that. However, the effect of independent PIs is not additive in this case. To wit, consider a two-machine, parallel machine makespan problem. Suppose that job p_j values are also indices, and that we start with the sequence 7-6-5-4-3-2-1. The makespan is 14, and it is optimal. Suppose we interchange jobs 5 and 4, then the makespan does not change. So the net effect of that PI is 0. Likewise, interchanging 1 and 2 has no effect. But performing both interchanges together leads to a makespan deterioration of 1. These two interchanges are independent by our current definition, but the effect of two independent interchanges is not additive ($0 + 0 \neq 1$). In effect, these so-called "independent" PIs are not entirely independent in the parallel machine environment. Hence, it would not be possible to apply the PI dynasearch neighborhood directly. More precisely, the DP at the heart of dynasearch would no longer be applicable with its present structure and complexity. Similarly, one might think that the PI dynasearch neighborhood could be applied to permutation flow shops with a makespan objective (see Chapter 10), but again the question is whether the effects of independent PIs are additive. If we define the effect of a PI as the change it induces in the makespan, it is easy to construct counterexamples with as few as two machines and four jobs showing that such

changes by independent PIs are not additive. Because job shops generalize flow shops, they too would be difficult to address. Finally, on the one hand, our pessimistic analysis here relates to the particular dynasearch neighborhood, not to the possibility of ever finding neighborhoods with exponential size that can be searched with polynomial effort. On the other hand, to our knowledge, all published exponential neighborhoods that can be explored in polynomial time relate to various single machine problems, such as the T_w-problem itself and the traveling salesperson problem (TSP), which we introduce formally in Chapter 8. (We also discuss it briefly next.)

Ant Colony and Neural Networks

We presented genetic algorithms as an approach that mimics nature. Likewise, simulated annealing was inspired by natural systems. Another nature-imitating heuristic approach that has been suggested for combinatorial optimization is the *ant colony* approach (Colorni et al., 1996). It is easiest to describe this approach for the TSP problem. The TSP requires a salesperson to visit n cities and return to base in the total shortest possible cycle. The TSP cycle can be specified by a permutation, prescribing the sequence in which the cities are visited (and such that the final connection is from [n] back to [1]). The associated decision problem is NP-complete in the strong sense. The source of the difficulty is that the total distance traveled depends on the sequence. (Nonetheless, as we already discussed briefly in our Research Notes for Chapter 3, the state-of-the-art TSP solver—Concorde—can handle very large instances effectively by a branch and cut approach.) Assume now that many ants start autonomously at various nodes that represent the cities and each solves the problem by itself, but it also knows where the other ants have been traveling most often. (In nature, ants mark their routes by scents—pheromones—so they can actually sense where other ants have traveled recently.) An ant selects the next city to visit at random, but it is more likely to select a near city and it is more likely to select a path that many other ants have used. The mechanism that emerges is a learning process that reinforces short and popular connections and eventually discards paths that are not used. So far, the evidence is that this approach works best when combined with other heuristics; e.g., one might add local search rules that enforce local optimality. With such enhancements, which are essentially tantamount to combining heuristics, ant colony is capable of achieving excellent results (den Besten et al., 2000). However, the computation time required for that purpose is several orders of magnitude higher than with other heuristics such as dynasearch; see Grosso et al., for a direct comparison between their dynasearch results and those of den Besten et al.

Yet one more heuristic approach that is based on nature involves the use of neural networks. By virtue of emulating the way a brain operates (at least as far as prevailing theories suggest), the potential of neural networks is promising (Agarwal et al. 2003, Agarwal et al., 2006, Colak and Agarwal 2005, and Gupta et al. 2000). Inherent similarities exist between the neural network and the ant colony models: both involve selection of strong connections that have been reinforced by previous experience.

Cross-Entropy

In general, the same heuristics that apply in deterministic scheduling can often be directly implemented for stochastic scheduling. But some heuristics are specifically

related to stochastic issues. In this connection, there are important stochastic approaches for deterministic problems and there are stochastic methods that are specifically relevant to stochastic models. The simplest example of a stochastic approach to a deterministic model is the use of random sampling. We also saw in the chapter that it is useful to bias the sampling process. More complex approaches include specific heuristics designed to perform biased random sampling effectively. This subject is also known as *importance sampling*. *Cross-entropy* (CE) is a strongly related approach. Suppose we select the best 10% of many randomly selected sequences and we observe that a particular job is first in a disproportionately large fraction of them. It then stands to reason that this job is in a sense a good candidate for the first position. We may therefore assign a higher probability for this job in biased sampling. The process could be repeated after running many instances based on this biased sampling, and so on. The CE approach is based on this idea. More details can be found in Rubinstein and Kroese (2004) and other references given below. The cross-entropy approach is also applicable to stochastic inputs in some cases. Using it for stochastic scheduling is a promising research area. For example, it has been used to estimate the probability of exceeding buffers in queueing networks, which is related to safe scheduling (de Boer et al., 2004).

Generating Dominance Properties for $1 \mid\mid \Sigma w_j T_j$

In this section we discuss precedence relationships for the basic T_w-problem. In the next section we consider the effect of *mandatory* precedence constraints, which extend the problem to one that has not yet received much attention in the literature, namely $1 \mid prec \mid \Sigma w_j T_j$. For distinction, we refer to dominance relationships as *optional* constraints, because we can choose to generate and impose them or to ignore them. Effective optimization algorithms work better with precedence constraints, so such results are potentially useful and we present them because they justify further research. Such research is needed not only for generating new ways to identify optional constraints but also for validating known relationships that are yet untested.

As discussed in our Research Notes for Chapter 3, Emmons (1969) provided the foundation for work on dominance relationships for the last four decades, for both unweighted and weighted cases. In the most recent work along these lines, Kanet (2007) methodically derived several such conditions, new and old. To describe that work, we start by reproducing some results from Chapter 3. In this section, we assume that the integers i and j satisfy $i < j$, but there is no predefined ordering between i and k or between j and k. Recall that $p(J)$ represents the sum of processing times in set J, X denotes the set of all jobs, A_j denotes all jobs known to succeed job j in at least one optimal sequence, $A_j' = X - A_j$ is the complement (which includes job j itself), B_k is the set of jobs that precede job k in at least one optimal sequence, and B_k' is the complement (so job $k \in B_k'$). Whereas it is convenient to use both types of sets, recall that A_j and B_k are mirror images of each other. Specifically, if job $k \in A_j$ then job $j \in B_k$. Conversely, if job $k \in A_j'$ then job $j \in B_k'$. In addition to jobs j and k, sets A_j' and B_k' include elements whose relationship with j and k (respectively) has not been resolved. The purpose of Theorem 3.3 is to enable moving job $k \neq j$ from A_j' to A_j. We reiterate that as we resolve such relationships it may become possible to resolve others that could not have been resolved initially. Our starting point is Theorem 3.3. Adding the conditions noted in the chapter, under which it can be extended to weighted problems, we obtain,

Theorem RN4.1. In the T_w-problem, there is an optimal schedule in which job k follows job j if one of the following conditions is satisfied:

(a) $w_j \geq w_k$, $p_j \leq p_k$, and $d_j \leq \max\{d_k, p(B_k) + p_k\}$;

(b) $w_j \geq w_k$, $d_j \leq d_k$, and $d_k \geq p(A_j') - p_k$;

(c) $d_k \geq p(A_j')$.

Condition (c)—which generalizes Theorem 3.2—simply states that if job k can be last in A_j' without tardiness, there is no benefit in scheduling it earlier, and thus there is an optimal schedule in which job k follows job j. Recall from our discussion in Chapter 3 that, in the unweighted case, condition (a) is a direct generalization of Theorem 2.8. Adding the condition that $w_j \geq w_k$ can only increase the benefit of following the prescribed order. For future reference, we note that unless $p_j/w_j \leq p_k/w_k$, condition (a) cannot be satisfied.

Kanet (2007) studies the possible ways in which each of three fundamental tactics can demonstrate that job k should follow job j, thus adding job k to A_j. We first discuss the first two tactics. For two jobs j and k with no known relationship, schedule job k at the end of the current set B_k (and thus, automatically, before job j, because we know that job $j \notin B_k$) and schedule job j just before the current A_j, at the very end of A_j'. Observe that there must be at least two jobs, including j and k, in $\{A_j' \cap B_k'\}$ (i.e., between the completion time of B_k and the start time of A_j), or we would already know how the jobs are related. Furthermore, for these two tactics there is no need to consider any schedule in which job k is scheduled earlier. Symmetrically, we need not consider any schedule in which job j starts later. Denote the set of jobs between k and j by H (i.e., $H = \{A_j' \cap B_k'\} \backslash \{j, k\}$). If H is empty, we can decompose our problem to three consecutive parts: B_k, $\{j, k\}$, and A_j. After solving each part separately, we obtain an optimal schedule. In such a case, we can sequence $\{j, k\}$ by trial and error or by (4.2) and thus also resolve whether $k \in A_j$ or B_j, but that resolution becomes purely academic. Partitioning the problem that way cannot but be advantageous as well. The more challenging assumption is that H is not empty. At this stage we can check whether job j is tardy. If job j is not tardy, then it is in A_k (this result is intuitive, but we prove it later). Otherwise, the first tactic studied by Kanet for this structure involves interchanging the two jobs. In this case, we assume $p_j \leq p_k$. The assumption is necessary to ensure that the effect on H will not be detrimental, thus allowing us to ignore it. The second tactic is inserting job k just after job j, which cannot be detrimental for H even if $p_j > p_k$. The moves used by both tactics decrease tardiness for job j but may increase it for job k. By comparing the cost of increasing tardiness for job k to the benefit of decreasing tardiness for job j, Kanet identifies sufficient conditions for allowing the move. A simple example is if $C_j \leq d_k$, so job k would not be tardy if inserted after job j and therefore we can safely add job k to A_j. That case boils down to Theorem RN4.1*c*. The third move starts with job j sequenced at any time before job k, but after $p(B_j)$ and before all the jobs in $\{A_j \cup A_k\}$—that is, job j must start before $p(X) - [p(A_j \cup A_k) + p_k]$. Here, sufficient conditions are sought for showing that inserting job k in any position before job j cannot improve our objective. If so, job $k \in A_j$. Now revisit the case where job j is not tardy at the end of A_j', where we claimed that

$\{j\} \in A_k$. If we interchange the names j and k, the two jobs start in the order we want to establish. Because the later job is not tardy, moving it to any earlier position, including any position before the earlier job, cannot improve the objective. So the third tactic proves the claim. Finally, once we add job k to A_j, by any relevant condition, all the jobs in A_k can now be considered also in A_j and all the jobs in B_k can be added to B_j.

As we indicated already, Kanet's results have not been tested yet, although some of them generalize or even repeat formerly known results that, as such, have been tested before. Therefore, to evaluate how powerful they are in reducing computation time and increasing the size of solved problems, we need empirical testing. When doing that, we should find not only whether all the new conditions are useful but also the best order for testing.*

It is possible to add mandatory constraints to the problem before starting the analysis. For instance, if job j must precede job k due to a mandatory constraint, we start the analysis with initial sets that satisfy $j \in B_k$ and $k \in A_j$. But doing so is not identical to just adding mandatory constraints to independently identified optional ones because optional constraints may change as a result of mandatory constraints; for instance, they may contradict each other.

We now present unpublished results of the same genre (also untested). Although these results can and should be treated as complements of published ones, we ignore most connections. To start, we reproduce Rachamadugu's test, (4.2), for two adjacent jobs, j and k, that start at time t. Job j can come first if,

$$\frac{p_j}{w_j}\left(1-\frac{s_k^+}{p_j}\right) \le \frac{p_k}{w_k}\left(1-\frac{s_j^+}{p_k}\right) \tag{4.2}$$

where s_i^+ is given by $(d_i - t - p_i)^+$; i.e., s_i^+ is the slack when positive, and zero otherwise.† Define the latest start time without tardiness of job i by $LS_i = (d_i - p_i)$. Then we may rewrite (4.2) as follows: job j precedes job k if they are adjacent to each other and,

$$w_j[p_k - (LS_j - t)^+] \ge w_k[p_j - (LS_k - t)^+]$$

Equivalently, define a difference function, $g_{jk}(t)$, such that if $g_{jk}(t) \ge 0$ then job j can precede job k when they are the next two jobs starting at time t. That is,

$$g_{jk}(t) = w_j[p_k - (LS_j - t)^+] - w_k[p_j - (LS_k - t)^+]$$

We refer to the relationship between two adjacent jobs, such as j and k, that start at time t and for which $g_{jk}(t) \ge 0$ as *stable*, and if all consecutive pairs are stable at their start time,

---

* See Exercise 2.6 but notice that here it is conceptually possible that one of the tests can predict several others, thus rendering them redundant. In Chapter 10, we discuss dominance tests for three-machine flow shops that exemplify this point.

† In general, sequencing algorithms that use slack are sometimes subject to problems associated with Theorem 2.7. But one way to ameliorate that effect is to use slack only if it is positive, and zero otherwise, as is the case here. With that structure in place, slack is not used for sequencing tardy jobs.

we say the sequence is stable. Suppose both jobs j and k can complete on time in either order (when they start at time t), then both orders are stable. Stability is really just an indication of local optimality, so we can pose a simple proposition,

Proposition RN4.1 At least one stable sequence is optimal.

By the proposition, the set of stable sequences is dominant, and from now on, we limit our attention to it. Both by the structure of (4.2) and by Theorem RN4.1*a*, SWPT is highly relevant to our problem. Define the *basic order* by SWPT, with ties broken by LS_j, and remaining ties broken by SPT (any further ties must be between essentially identical jobs and may be broken arbitrarily). Without loss of generality, we henceforth assume that all jobs are indexed by the basic order. If jobs k and k+1 are identical, we can arbitrarily set $(k + 1) \in A_k$ (and thus also $k \in B_{k+1}$), without risk of suboptimality. Recalling that $i < j$, by (4.2), it is clear that if jobs i and j are considered for scheduling next to each other and i is already tardy, then i can come first (the left element is at most p_i/w_i and the right at least p_j/w_j so the ordering is assured by SWPT). One consequence is that it is sufficient to define $g_{ij}(t)$ for $0 \le t \le LS_i$. Beyond LS_i the function would be nonnegative; i.e., in such a case, we can safely schedule by the basic order. For that reason, for any pair of potential adjacent jobs there exists a time, v_{ij} ($\le LS_i$), such that for any $t \ge v_{ij}$ job i can precede job j when they are adjacent. (In the rare event that, for two non-identical jobs, $p_i/w_i = p_j/w_j$ and $LS_i = LS_j$, the tie-breaker in the basic order definition implies that $p_i < p_j$. In such a case we obtain $v_{ij} = 0$. We specified that tie-breaker for this reason.) We can also decree that job i precedes job j if neither of them will be tardy as a result. That is, when no tardiness is involved, we refer only to the order $i \rightarrow j$ as stable. Proposition RN4.1 remains valid after this restriction on the set of stable sequences: that is, the best stable sequence in the restricted set is optimal.

Rachamadugu used (4.2) recursively to demonstrate that if all jobs are tardy when sequenced by SWPT, then SWPT is optimal. Because the argument is recursive, if it applies to the first k jobs but not later, we can still sequence the first k jobs first. Effectively, then, we can remove them from the problem. As explained in Chapter 3, such removal requires adjusting all due dates by subtracting the total processing time of these k jobs; we also reduce the index of all remaining jobs by subtracting k. Thus, without loss of optimality, we assume that we checked these conditions in a preprocessing step and therefore no such jobs exist. That is, $LS_1 > 0$. It is possible in an optimal sequence for a high-index job, say j, to directly precede a low-index job, say i, even if it causes tardiness, but only up to a limit. That may happen if by scheduling job j first, we avoid large tardiness in this job at the expense of a sufficiently smaller tardiness in job i. When that is the case, we must have $g_{ij}(t) < 0$ for some $0 < t < LS_i$. If so, $LS_i + \max\{p_j\}$ is an upper bound on the start time of job i, when it follows a job with a higher index. We can calculate sharper bounds by explicitly considering all possible preceding jobs, j, as we discuss next.

By inspection, it is clear that $g_{ij}(t)$ is a piecewise linear function with a breakpoint at LS_j. For our purpose, this breakpoint is only important if $0 < LS_j < LS_i$. If so, s_j^+ becomes zero there, whereas s_i^+ remains positive. Also, $g_{ij}(LS_i) \ge 0$ (by SWPT, because i

$< j$). To simplify our presentation, we henceforth assume LS_i, $g_{ij}(LS_i) > 0$, but our conclusions remain intact if $LS_i \leq 0$ (which is operationally equivalent to $LS_i = 0$) or $g_{ij}(LS_i) \geq 0$. Because $g_{ij}(t)$ is piecewise linear, $g_{ij}(t) = 0$ is only possible for an argument t in the range we consider if $g_{ij}(0) < 0$ or $g_{ij}(LS_j) < 0$ (or both). Furthermore, if $g_{ij}(LS_j) < 0$, then $LS_j < LS_i$ and an argument t such that $g_{ij}(t) = 0$ must exist such that $LS_j < t < LS_i$, and the derivative of $g_{ij}(t)$ is positive there. If $g_{ij}(0) < 0$, the same observation applies but, if $LS_j < LS_i$ and $g_{ij}(LS_j) > 0$ the argument t resides between 0 and LS_j. In both cases there is exactly one such argument and the derivative is positive at this argument. If $g_{ij}(0) > 0$ and $g_{ij}(LS_j) > 0$ or $LS_j \geq LS_i$, no such argument exists. Recall that we defined v_{ij} such that after $t \geq v_{ij}$ job j should not precede job i directly. We now see that either $v_{ij} = 0$ or it is given by the argument t for which $g_{ij}(t) = 0$ and the derivative is positive. Now consider the special case where $g_{ij}(0) > 0$ but $g_{ij}(LS_j) < 0$ (which also implies $LS_j < LS_i$). In this case, $v_{ij} > LS_j$ but between 0 and LS_j there must be another argument t for which $g_{ij}(t) = 0$, such that $g_{ij}(t)$ is decreasing in the neighborhood. When such a value exists, we call it u'_{ij}. Otherwise, we define $u'_{ij} = 0$. Our definitions satisfy the following conditions:

(i) $0 \leq u'_{ij} \leq v_{ij} \leq \max\{0, LS_i\}$
(ii) if $u'_{ij} > 0$ then $g_{ij}(u'_{ij}) = g_{ij}(v_{ij}) = 0$ and $u'_{ij} < v_{ij}$
(iii) $g_{ij}(t) \leq 0$ if and only if $0 \leq u'_{ij} \leq t \leq v_{ij} \leq LS_i$.

Recall from previous discussion that if both jobs can be on time in basic order, that is, if $t \leq \min\{d_i - p_i, d_j - p_i - p_j\}$, we prefer to perform job i first. To reflect that we now define the value u_{ij} as the earliest time for which we must consider the possibility that j can precede i and still comply with Proposition RN4.1. We calculate it by

$$u_{ij} = \max\{\min\{d_i - p_i, d_j - p_i - p_j\}, u'_{ij}\}$$

Computing u_{ij} and v_{ij} for all $i < j$ is straightforward. We can store all these values in advance in two $n \times n$ upper triangular matrices, or in a single full matrix where we store the v_{ij} values above the diagonal and the u_{ij} values below the diagonal. We refer to that matrix as the *stability matrix*. By Proposition RN4.1, we restrict ourselves to sequences where i always precedes j when they are adjacent unless $u_{ij} < t < v_{ij}$. When $v_{ij} = 0$, job j needs not be considered directly before job i in any schedule.

We can now strengthen Rachamadugu's observation that if jobs are tardy under SWPT then SWPT is optimal. Consider job 1 in the basic order. If all $v_{1j} = 0$, then job 1 must be first, or it would have to follow a job that should, or at least could, have followed it instead. In such a case we can schedule job 1 first and remove it from further consideration. Furthermore, suppose the relationships among the first k jobs are not clear but for any $i \leq k < j$, $\min_i\{v_{ij}\} = 0$, then jobs 1, 2, . . ., k should be scheduled first, in some order. In that case, the problem is effectively decomposed to scheduling the first k jobs and the next $n - k$ jobs afterwards. Similarly, suppose $v_{jn} = 0$ for all j, then job n cannot be stable anywhere except in the last position and it can be scheduled last and removed from further consideration. Furthermore, job 1 may only follow job k directly if job k starts at some time t that satisfies $0 \leq u_{1k} \leq t < v_{1k} < LS_1$. Let $V(m)$ denote the set of jobs $k > m$ for which $v_{mk} > 0$. Therefore $T_1 \leq \max_{k \in V(1)}\{v_{1k} + p_k - LS_1\} < p_k$ (for the same k). A similar expression applies for job 2, with one exception: job 2 may have to follow job 1

even if it causes higher tardiness in job 2 than would be allowed if job 2 were to follow a job with a higher index. Therefore, $T_2 \leq \max\{\max_{k\in V(1)}\{v_{1k} + p_k + p_1 + p_2\}, \max_{k\in V(2)}\{v_{2k} + p_k + p_2\}\} - d_2$. Job 3, in turn, also has a similar limit on its tardiness, but in this case with two exceptions: it may have to follow job 2 (regardless of whether job 2 follows job 1) and it may have to follow job 1. By such analysis we can bound the maximum completion time of all jobs, recursively. This is potentially useful as long as the bounds are below $p(X)$. Furthermore, we can create analogous bounds on the earliest start time of jobs n, $(n - 1)$, $(n - 2)$, etc. Let $U(m)$ denote the set of jobs $k < m$ for which $v_{km} > 0$. A schedule in which job n starts before $\min_{k\in U(n)}\{u_{kn}\}$ cannot satisfy the stability requirement and may not be considered. Job $n - 1$ may precede job n directly in some stable sequences, but otherwise it should not start before $\min_{k\in U(n-1)}\{u_{k,n-1}\}$, etc. If for some integers a and b the earliest start time of job $(n - b)$ exceeds the latest completion time of job a, then jobs 1, 2, . . ., a all precede jobs $(n - b)$, $(n - b + 1)$, . . ., $(n - 1)$, n. Furthermore, for pairs of jobs that do not quite satisfy that condition, such as job a and job $(n - b - 1)$, or job $(a + 1)$ and job $(n - b)$, the bounds may still be useful for tests associated with either one of the first two tactics defined by Kanet. In such case, their role is equivalent to the roles of B_k and A_j' in restricting the times at which jobs j and k are scheduled.

At least for presentation purposes, assume that the problem is solved by branch and bound, scheduling forward.* Thus, at some stage of the scheduling process we will have a partial schedule, PS, which is a set of jobs scheduled to start at time 0 and complete at time $p(PS)$. The number of potential branches following PS can be reduced if we construct a set of precedence relationships for the set of schedulable jobs, $\{X - PS\}$. Furthermore, we know with certainty that an optimal sequence exists that is stable. Therefore, we can restrict our attention to compliant sequences. For efficiency, it is desirable to be able to perform all the necessary calculations for this task at time 0, so that we will not have to regenerate conditions for every possible PS. The stability matrix can serve this purpose very well. We just limit our attention to unscheduled jobs and we subtract $p(PS)$ from each u_{ij} and v_{ij} value and set them to zero if they become negative. One way to proceed is to branch first on jobs 1, 2, . . ., k subject to the condition that they can be directly followed by some job. For job k this implies that there exists at least one value $j > k$ such that $v_{kj} \leq p_k$, or one value $j < k$ such that $u_{jk} \leq p_k \leq v_{jk}$. When no such j exists, job k cannot be first.

We explored the efficacy of these insights by incorporating them into a heuristic procedure. The first step, starting with job 1, is to schedule as many consecutive jobs as possible without tardiness. This can be done in EDD order. Suppose we fit k jobs in this manner, and therefore job k+1 is either tardy itself or causes tardiness downstream. We know that job k+1 can fit right after the previous k jobs, in which case it will be tardy but it will follow a job with a lower index (which can be stable). But we may also try to insert job k into the latest position where it can be on time (it cannot be stable earlier), or later. If the total weighted tardiness decreases by such an insertion, we adopt the best one and move on to job k+2, etc. The worst case complexity of the heuristic is $O(n^2)$; i.e., a very low polynomial. To see this, suppose we already scheduled m jobs and our task is either to place the next job in its correct place by EDD (if there is no tardiness yet)—which we

* Readers may wish to compare our structure to that of Algorithm 14.1, which applies to the more general job shop (page 334).

can choose to treat as an insertion—or to insert the next job in the best position possible (after tardiness manifests). Place job m+1 in the last position and instead of direct insertion move it earlier by a series of $O(m)$ APIs. Each API takes constant time and the combined effect of the series is given by the sum of the individual effects. As we go along, we can also note the best position identified so far, so later we can return to it directly. As we have to repeat this procedure n times, and $O(m) = O(n)$, the total complexity is $O(n^2)$. We tried the heuristic on the two test problems that fared worst under WMDD. It cut the suboptimality in one from 10% to 4% and in the other from 7% to less than 1%. On the one hand, there is no need for such a heuristic as we can achieve better results by basic neighborhood searches, as well as advanced ones (such as dynasearch). On the other hand, a fair assessment of the heuristic would pitch it only against $O(n^2)$ or better alternatives, and in this arena its performance seems promising. Nonetheless, our testing was not sufficient for drawing firm conclusions so this question requires further research. Having said that, recall that we were not primarily motivated to find a new heuristic solution but rather we aimed to explore the efficacy of our new insights. Thus, the most important conclusion of this analysis is that it reinforces the notion that local stability conditions can be used to achieve good results for the whole sequence.

We note again that job 1 can be scheduled first if $v_{1j} = 0$ at time 0 for all j. This test can also be carried out for the unscheduled job with the lowest index as part of an optimizing algorithm. As branching progresses, we may find that for a particular branch (and thus a particular *PS*) the next few jobs in basic order can safely be added to the branch serially. This event is actually quite likely in problems where the due dates are such that it is clear that quite a few jobs must be tardy. Another likely event is that no unscheduled job can be appended to *PS* at the end and be stable with respect to the last job in *PS*. Such a branch can be fathomed. Similar observations may apply to jobs in the last positions, but they would likely be more important for scheduling backwards. Only experimentation can reveal, however, if checking such conditions is beneficial. Furthermore, we just explored the tip of the iceberg in terms of opportunities to develop (4.2) to more general dominance properties. To clarify, Kanet (2007) probably exposed the most useful relationships that can be developed for two jobs j and k that have partial sets of known predecessors and followers and are scheduled appropriately. As noted before, it is likely that the jobs will be separated by a set (denoted H) of intermediary jobs. Recall that it was necessary to represent the effect of an exchange on H by a bound. Developing (4.2), however, is based on the idea that we may be able to characterize conditions under which the jobs in H actually encourage exchanging jobs j and k or where we can show that some of the jobs in H could (and should) be added to the given sets of predecessors and successors instead.

Adapting Search Heuristics to the Solution of $1 \mid prec \mid \Sigma w_j T_j$

Next, we consider a generalized version of the T_w-problem with precedence constraints. This problem is currently open. The two versions, with and without precedence constraints, are closely related because we can always choose to treat optional relationships as mandatory. However, the presence of mandatory constraints may change the calculations and details involved in identifying optional constraints. For example, take the unweighted case. It may be that a job that gets a low priority and is assigned a

late position under the conditions of Theorem RN4.1 is the predecessor of a job that would, by itself, acquire an earlier position. Then it is likely that the priority of the predecessor job should be increased, but it is also likely that the successor will be delayed. Some of the existing relationships can still be derived using mandatory constraints to start the process of adding new relationships. However, the results we developed above by studying (4.2) require modification or may not work with mandatory constraints. Nevertheless, a more important question in practice is whether neighborhood search techniques that currently provide the best practical approach to the T_w-problem can be adapted to $1 \mid prec \mid \Sigma w_j T_j$, and if so, how? In this section we propose ways to make such heuristics avoid searching infeasible sequences without losing the opportunity to find the optimal solution. If future experience demonstrates that search heuristics, such as dynasearch, can benefit from incorporating precedence constraints, mandatory or optional, then we can choose to include them even when they are all optional.

In Chapter 17 we present a modified API search that maintains precedence constraints and is thus applicable for any API neighborhood search for the T_w-problem with constraints. The general idea is simple: if we try to move a job (called an *activity* in the project context) to a position earlier than a predecessor, we may also have to move its predecessors. If a set of precedence relationships has been identified by previous analysis, then we treat them as hard constraints. The modified API approach can be generalized for the insertion neighborhood without any conceptual difficulty. We can also define a modified PI between unrelated jobs as a double insertion. Suppose that we want to interchange jobs i and k, currently in that order. We can safely assume that these jobs are unrelated, or the interchange can simply be aborted. Let G denote the set of jobs that precede job i, let H denote the set of jobs strictly between jobs i and k, and let J denote the jobs that follow. In other words, the current sequence is G-i-H-k-J and without precedence constraints, a PI would involve a move to G-k-H-i-J. When constraints are imposed, we still restrict modified insertion or modified exchange to the set $\{i, H, k\}$, leaving both G and H scheduled as before. If any job in $A_i \cup A_k$ resides in J, our restriction to $\{i, H, k\}$ guarantees that it will still follow its predecessor. A symmetric observation holds for any jobs in $\{B_i \cup B_k\} \cap G$. Consider the (possibly empty) subset of jobs in H that belong to A_i, namely $A_i \cap H$, and the subset of jobs in $B_k \cap H$, the known predecessors of k within H. These two subsets must be unrelated in the sense that no job in the former is a predecessor of any job in the latter (or jobs i and k would be related too). The modified PI can start by inserting the jobs in $\{B_k \cap H, k\}$—in their original order—just after G (thus automatically forbidding k from taking the first position after G unless $B_k \cap H$ is empty). Another, similar, insertion step moves the jobs $\{i, A_i \cap H\}$ to a position just before J, again maintaining the original order of jobs within $A_i \cap H$, with job i preceding the others. The result is a legal modified PI that is guaranteed to be feasible if jobs i and k are unrelated. Another point, relevant to dynasearch, is that after this exchange, any set of exchanges strictly within G or strictly within J would be independent from the change we just described within $\{i, H, k\}$. Thus, it is conceptually possible to address $1 \mid prec \mid \Sigma w_i T_i$ by modified dynasearch (with the basic or the enhanced neighborhood definition). The open question is how efficient and effective such a search may be. If the dominance conditions actually make the search easier (and they certainly reduce the number of feasible sequences, which should help), then it may be useful to generate them when they are

optional. Otherwise, we still need to account for mandatory constraints, and we just demonstrated how that might be done.

Finally, it is interesting to study the efficacy of GA in the presence of precedence constraints. As it turns out, GA is inherently adapted to incorporating precedence constraints, but we have to be careful with mutations. Recall that under the standard method of generating offspring, except for mutations, they maintain the job sequence of at least one parent. Therefore, if two parents obey the constraints, so do their offspring. However, general mutations can violate precedence. Thus, in the case of GA, it might be useful to adapt the structure of mutations so that they will not violate constraints. Such adaptation essentially boils down to limiting mutations to modified moves.

Asymptotic optimality

A heuristic is *asymptotically optimal* if, as n grows large, the relative difference between the heuristic solution and the optimum becomes negligible. More formally, let $f(S^*)$ denote the objective function value with the optimal sequence, S^*, and let $f(S^H)$ be the value associated with a heuristic. We say that the heuristic is asymptotically optimal if, in the limit as $n \to \infty$, $[f(S^H) - f(S^*)] / f(S^*) \to 0$. When we can prove that a heuristic is asymptotically optimal, then in a sense we can say that it is a good heuristic. Such proofs typically require regularity conditions on job characteristics, however. We are especially interested to know whether a relatively simple construction heuristic meets that test. If such a heuristic is asymptotically optimal, then we can typically solve large problems to near-optimality, while we can address medium problems by supplementing the construction heuristic with neighborhood search techniques and obtain optimal or near-optimal results. Small problems are inherently easier, so in a practical sense we can say that such a problem is solved for any size. We discuss several instances of asymptotically optimal heuristics later in the text, both for deterministic and stochastic problems. However, we do not know of any existing asymptotically optimal heuristic for the tardiness problem (which is one of the reasons it is a good problem to pursue in our first chapter on heuristics). In particular, the following example demonstrates a case where the MDD heuristic is far from asymptotically optimal, at least when no regularity conditions are imposed: instead, as $n \to \infty$, $[f(S^H) - f(S^*)] / f(S^*) \to n / 2$. The source of this example—Della Croce et al. (2004)—provides similar examples designed to expose the weaknesses of other popular heuristics as well, so this should not be interpreted as specific criticism of MDD.

Example RN4.2 Consider an $(n + 1)$-job single-machine minimal tardiness instance with $p_1 = d_1 = n$; $p_2 = \ldots = p_{(n+1)} = 1$; $d_2 = \ldots = d_{(n+1)} = n + \varepsilon$, where ε is a strictly positive value but as small as we may wish (and we will assume ε is infinitesimal).

Recall that we define the modified due date of job j at time t to be

$$d_j' = d_j'(t) = \max\{d_j, t + p_j\}$$

In the example, for $t < \varepsilon$, job 1 has the minimal modified due date; e.g., $d_1'(0) = n$ whereas $d_j'(0) = n + \varepsilon$ for any $j > 1$, so job 1 will be selected as the first job. The total

tardiness in this case is $n^2 / 2 - n /2 - n\varepsilon /2$, instead of the optimal total tardiness of n, obtained by placing job 1 last. For large n and infinitesimal ε, the *error ratio* is $n /2$, which is unbounded. The suboptimal solution would remain unchanged even if we were to conduct an API neighborhood search, but it would be corrected with any of the more advanced search methods that we presented. The potential failure of API in this case is not surprising because MDD always yields a solution that is locally optimal in the API neighborhood. Perhaps the most important practical conclusion from Example RN4.2 is that combinations of heuristics tend to be more robust than any single heuristic.

Testing Heuristics

In spite of the existence of fabricated worst-case examples that yield such disappointing results, there is no question that heuristics are necessary. For that reason, researchers and practitioners in sequencing and scheduling need to be able to develop, test and apply heuristics. For most practitioners, the aim is not originality but effectiveness. Hence, they can select from the existing arsenal the best heuristic or combination of heuristics for the actual conditions they wish to address. In doing so, they may prefer to use off-the-shelf software (e.g., the Evolutionary Solver). Here, however, we focus more on developing and testing new heuristics as a research endeavor. Just inventing a new heuristic is only the beginning. The next task is to demonstrate that the heuristic is in some sense sufficiently superior to justify using it instead of, or in addition to, existing solutions. This requires extensive testing. A key question is the quality of the test problems on which the heuristic is tested (Baker 1999). This is the point at which many efforts founder.

To test and compare heuristics for a particular practical application, we may be able to use realistic data sampled from the production floor. To the extent that the sample is representative not only of the present but also the future, the result is then good for the application. But for a heuristic to be considered competitive in general, it should be tested on instances known to be difficult. Test conditions where the problem is trivial should be avoided. For instance, if we use problems with agreeable due dates and processing times even the simplest heuristic (say EDD or SPT) will yield optimal results. Instead, we should attempt to find conditions under which the solution procedure is most severely challenged. Accordingly, in the chapter we used test problems that are known to be difficult (relative to their size). Indeed, those problems were developed as part of research into the question what constitutes a difficult instance for the weighted tardiness model. Thus we can be relatively confident that the heuristics will work reasonably well in practice, too. Hall and Posner (2001) discuss the issue of selecting non-trivial test sets for any objective. Here we elaborate a bit on the difficulty of various instances of the T-problem.

Early results about this issue appeared in Srinivasan (1971), Wilkerson and Irwin (1971) and were later refined by Baker and Martin (1974). Two insights motivate these results. We might guess that the performance of a heuristic depends on how many jobs are likely to be tardy. If no jobs are tardy, we can produce the optimal solution with EDD sequencing. If all jobs must be tardy, we can produce the optimal solution with SPT sequencing. Thus, we might expect that problems are most difficult to solve when some, but not all, of the jobs are likely to be tardy. A simple way to operationalize that notion is to say that problems are most difficult when the due dates, on average, lie neither at the

beginning nor the end of the schedule. A second insight relates to the dispersion of the due dates. If they are spread widely around a given average, then it may be easier to sequence jobs so that individual due dates are met, as compared to a case where the due dates are clustered. We start by describing how we might create samples reflecting some of these predefined characteristics, and then we discuss the resulting difficulty of the results.

Define the *tardiness factor*, denoted *t*, 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 samples are taken and let μ_d denote the mean due date. Therefore, $t = 1 - (\mu_d / n\mu_p)$. For a desired level of *t*, set

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

In other words, we first decide on a processing time distribution and choose its mean. Then, for some desired level *t*, we calculate μ_d and sample due dates from a distribution with that mean. Next, we define the *due-date range*, denoted *r*, as the range of the due dates relative to the makespan. Again, for the purposes of generating data, we might sample due dates from a uniform distribution on the interval (*a*, *b*). This implies

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

Thus, we decide on a mean processing time, and for a desired value *r*, we calculate the width of the range, $(b - a) = rn\mu_p$. Knowing *t* (above) gives us the mean of the due date distribution, so once we know the width of the range, the uniform distribution is fully specified, and we can draw samples. Difficult problems—like the ones we have used in the chapter—involve tardiness factors of roughly 0.6 to 0.8, along with a tight due date range of 0.2.

We can see now that there is no conflict between the fact that Example RN4.2 is one for which MDD does not work well and that it performed well for a set of test problems developed according to such principles. The key to the difference is that the test problems were generated according to principles but randomly, whereas a cursory examination of Example RN4.2 reveals that it is not likely to be the result of a random selection. Asymptotic optimality theorems typically invoke regularity conditions that would rule out Example RN4.2. However, we repeat that MDD has not been proven asymptotically optimal even when subject to such regularity conditions.

More on Asymptotic Behavior

We defined asymptotic optimality for problems with a given number of jobs, *n*. In a dynamic application where jobs arrive at random times on an ongoing basis, a more common term is *asymptotic convergence*. The word "convergence" suggests some infinite process, which is indeed the case in such a dynamic environment. However, asymptotic convergence is also used for another type of desirable asymptotic behavior for problems with a given number of jobs, *n*. We say that a search heuristic converges asymptotically if we can reach an optimal solution by running the search for a sufficiently long time (*w.p.*1). That is, as the number of iterations approaches infinity, we will have identified an optimal solution almost surely (but we will not have any proof that this

solution is indeed optimal). In contrast to asymptotic optimality, this definition makes sense only for search heuristics and not for single-pass construction heuristics: the search itself constitutes the potentially infinite process that is implied by the term "convergence." Whereas asymptotic optimality asks whether a heuristic's relative error becomes negligible for large enough instances (and typically applies to construction heuristics), asymptotic convergence asks whether a search heuristic is liable to get stuck forever at a local optimum (regardless of instance size). For example, API search is not asymptotically convergent because it can get stuck this way. By contrast, random search is asymptotically convergent, even when the sampling is biased (as long as all p_{kj} are strictly positive so every single permutation has a positive probability of being selected). Other heuristics that involve random elements and large enough neighborhoods, such as simulated annealing (with a sufficiently slow cooling regime), are also asymptotically convergent: given enough time the search will stumble on an optimal sequence. Genetic algorithms employ mutations for this purpose, but they do not guarantee asymptotic convergence (Ingber and Rosen, 1992). Nonetheless, without exception, any neighborhood search heuristic that is used repetitively as per the multi-start policy with randomized seeds is convergent.

Concluding Remarks

Even among the heuristics that are competitive today, it is difficult to assess which approach is "best." Refinements keep being developed and the target—that is, the set of problems for which such heuristics are tested—keeps moving. (The examples we used in the chapter were selected for pedagogical reasons, but they are no longer even close in complexity to the ones current research addresses.) In this connection, we should highlight a distinction between using heuristics well and reporting their merits and shortcomings well. There is little doubt that heuristics can and often should be combined, sometimes with great synergy. At the very least, running two or more distinct but sufficiently fast heuristics and selecting the best result cannot hurt. For instance, just by the evidence provided by the two examples we solved, if we were to combine the WMDD heuristic with the new heuristic we presented above, the average deviation would drop from 2% to 1% or less, and the maximum, from 10% to 4%. Furthermore, practically all optimization algorithms involve internal heuristic choices that can influence their performance significantly. For instance, the simplex algorithm chooses the candidate entering variable that improves the objective function fastest (on a per unit basis). But we could also select the candidate variable that improves the objective function most in the next step. This rule often leads to the same candidate, but not always. Both rules are essentially greedy heuristics, and it took experimentation to decide which one to use. Thus, even in developing an optimization platform, we must test and use heuristics. Indeed, there is an emerging approach called *variable neighborhood search* (VNS) that essentially tries several neighborhoods for each move and selects the best. That creates a single *metaheuristic*, reportedly a highly successful one (Hansen and Mladenović, 2001; Hansen et al., 2006). Hence, combining heuristics is often a productive idea, although selecting a good combination is an art. However, the picture is less clear with respect to reporting results. There are many potential combinations, and each of them involves many seemingly minor implementation decisions that may not be minor in fact. Therefore, it would be counterproductive to make broad comparisons

among such combinations. When a combination of heuristics works well, it may be impossible to tell which ingredients should really be credited. For example, in Chapter 14 we discuss a highly successful state-of-the-art tabu search algorithm for job shops (due to Nowicki and Smutnicki, 2005) that involves so many refinements and clever bound calculations that we can no longer say with certainty how much of the success is due to the tabu search mechanism, or any other ingredient. (For that reason, when we performed experiments to evaluate some of the heuristics, we restricted our comparisons to "vanilla" applications of the individual heuristics.)

With this caveat, three heuristic approaches are frequently mentioned as winners in more complex environments: tabu search, simulated annealing and genetic algorithms. Of these, TS is reportedly very competitive for job shop makespan minimization, closely followed by SA (Vaessens et al. 1996). But for projects, there is evidence that GA may be the best choice (Hartmann 2001). Indeed, different approaches may be required for different problems in general. Although such results cannot be taken as "final" or completely objective, there are possible explanations why one approach is better in the project environment and another in the job shop. Specifically, in the job shop environment, it is easier to confine the search to feasible solutions that are likely to improve upon the current candidate than it is in the project environment. Therefore, because GA is likely to produce conforming offspring, it has an edge for projects. We suspect that it would perform even better if mutations were to be modified. If so, however, then there is room to improve the other approaches for projects by modified neighborhoods. Indeed, Fleszar and Hindi (2004) provide partial empirical evidence to that effect in the project scheduling context. They use an insertion neighborhood that conforms to the modified insertion mechanism we introduced. The heuristic is reportedly very successful, but it also involves variable neighborhood search. Thus, again, it is difficult to judge how much of their success is attributable to VNS and how much to the use of modified search moves. We discuss this point further in the research notes of Chapters 14 and 18.

Optimal Values of the 12 Test Problems

For readers who might wish to try solving the 12 test problems, we list the optimal target function values here.

#	*T*	#	*T*	#	*T*
1	78028	5	32370	9	102709
2	116674	6	47542	10	40232
3	69558	7	40067	11	47780
4	32992	8	85800	12	49704

An Excel file containing the problem data can be found among the Data Files on the book's website.

References

Agarwal A., Colak S., Eryarsoy E. (2006) "Improvement heuristic for the flow-shop scheduling problem: An adaptive-learning approach," *European Journal of Operational Research* 169, 801-15, 03 2006.

Agarwal A., Pirkul H., Jacob V.S. (2003) "Augmented neural networks for task scheduling," *European Journal of Operational Research* 151, 481-502.

Baker, K.R. (1999) "Heuristic Procedures for Scheduling Job Families with Setups and Due Dates," *Naval Research Logistics* 46, 978-991.

Baker, K.R. and J.J. Kanet (1984), "Improved Decision Rules in a Combined System for Minimizing Job Tardiness," *International Journal of Production Research* 22, 917-921.

Baker, K.R. and J.B. Martin (1974) "An Experimental Comparison of Solution Algorithms for the Single-Machine Tardiness Problem," *Naval Research Logistics Quarterly* 21, 187-199.

de Boer, P.T., D.P. Kroese and R.Y. Rubinstein (2004) "A Fast Cross-Entropy Method for Estimating Buffer Overflows in Queueing Networks," *Management Science* 50, 883-895.

den Besten, M., T. Stützle and M. Dorigo (2000) "Ant Colony Optimization for the Total Weighted Tardiness Problem," *Parallel Problem Solving from Nature*, Lecture Notes in Computer Science, Volume 1917, 611-620, Springer Berlin/Heidelberg.

Colak S., Agarwal A. (2005) "Non-greedy heuristics and augmented neural networks for the open-shop scheduling problem," *Naval Research Logistics* 52, 631-44.

Colorni, A. M. Dorigo, F. Maffioli, V. Maniezzo, G. Righini and M. Trubian (1996) "Heuristics from Nature for Hard Combinatorial Optimization Problems," *International Transactions in Operational Research* 3, 1-21.

Congram, R.K. (2000) Polynomially searchable exponential neighborhoods for sequencing problems in combinatorial optimisation, Ph.D. thesis, University of Southampton, UK. (Available on the web.)

Congram, R.K., C.N. Potts and S.L. van de Velde (2002) "An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem," *INFORMS Journal on Computing* 14, 52-67.

Della Croce, F., A. Grosso and V. Th. Paschos (2004) "Lower Bounds on the Approximation Ratios of Leading Heuristics for the Single-Machine Total Tardiness Problem," *Journal of Scheduling* 7, 85-91.

Fleszar, K. and K. Hindi (2004) "Solving the Resource-Constrained Project Scheduling Problem by a Variable Neighbourhood Search," *European Journal of Operational Research* 155, 402-413.

Glover, F. (1989) "Tabu Search—Part I," *ORSA Journal on Computing* 1, 190-206.

Glover, F. (1990a) "Tabu Search—Part II," *ORSA Journal on Computing* 2, 4-39.

Glover, F. (1990b)"Tabu Search: A tutorial," *Interfaces* 20(4), 74-94. (Available at: http://leeds-faculty.colorado.edu/glover/TS%20-%20Interfaces.pdf [accessed February 22, 2009].)

Grefenstette, J.J. (ed) (1985) *Proccedings of the 1st International Conference on Genetic Algorithms*, L. Erlbaum Associates Inc. Hillsdale, NJ, USA.

Grosso, A., F. Della Croce, R. Tadei (2004) "An enhanced dynasearch neighborhood for the single-machine total weighted tardiness scheduling problem," *Operations Research Letters* 32, 68-72.

Gupta J.N.D., Sexton R.S., and Tunc E.A. (2000) "Selecting scheduling heuristics using neural networks," *INFORMS Journal on Computing* 12, 150-62.

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

Hansen, P. and N. Mladenović (2001) "Variable Neighborhood Search: Principles and Applications," *European Journal of Operational Research* 130, 449–467.

Hansen, P., N. Mladenović and D. Urošević (2006) "Variable Neighborhood Search and Local Branching," *Computers & Operations Research* 33, 3034–3045

Hartmann, S. (2001) "Project Scheduling with Multiple Modes: A Genetic Algorithm," *Annals of Operations Research*, 102, 111-135.

Holland, J.H. (1975) *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor.

Ingber, L. and B. Rosen (1992) "Genetic algorithms and very fast simulated reannealing: a comparison," *Mathematical and Computer Modelling*, 16(11), 87-100.

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.

Kirkpatrick, S., C.D. Gelatt and M.P. Vecchi (1983) "Optimization by Simulated Annealing," *Science* 220, 671-680.

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.

Morton, T.E. and D.W. Pentico (1993) *Heuristic Scheduling Systems*, Wiley, New York.

Nowicki, E. and C. Smutnicki (2005) "An Advanced Tabu Search Algorithm for the Job Shop Problem," *Journal of Scheduling* 8, 145-159.

Pan, Y. and L. Shi (2007) "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 S. Van de Velde (1995) "Dynasearch—iterative local improvement by dynamic programming: part I, the traveling salesman problem," Technical Report, University of Twente, The Netherlands.

Rubinstein, R.Y, and D.P. Kroese (2004) *The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation and Machine Learning*, Springer.

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.

Vaessens, R.J.M., E.H.L. Aarts and J.K. Lenstra (1996) "Job Shop Scheduling by Local Search," *INFORMS Journal on Computing* 8(3), 302-317.

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

A tutorial on the cross-entropy method is available at the CE web page, http://www.cemethod.org and also at, http://wwwhome.cs.utwente.nl/~ptdeboer/ce/tutorial.pdf. Accessed: 24 Oct. 2007)