Local Search in Combinatorial Optimization
Edited by E. Aarts and J. K. Lenstra

11

Machine scheduling

Edward J. Anderson
University of New South Wales, Sydney
Celia A. Glass, Chris N. Potts
University of Southampton, Southampton

1 INTRODUCTION

The scheduling of computer and manufacturing systems has been the subject of extensive research since the early 1950s. The range of application areas for scheduling theory goes beyond computers and manufacturing to include agriculture, hospitals, transport, etc. The main focus is on the efficient allocation of one or more resources to activities over time. It is convenient to adopt manufacturing terminology: we refer to a *job* which consists of one or more activities, and a *machine* which is a resource that can perform at most one activity at a time.

We restrict our attention to *deterministic machine scheduling* where it is assumed that the data that define a problem instance are known with certainty in advance. An excellent survey of the area is the paper by Lawler et al. [1993], and the textbooks of Conway, Maxwell & Miller [1967], Baker [1974], French [1982] and Błazewicz et al. [1994] provide an introduction.

Much of the early work on scheduling was concerned with the analysis of single-machine systems. Examples include Jackson's derivation of the earliest due date (EDD) rule in which jobs are sequenced in order of nondecreasing due dates [Jackson, 1995], and Smith's derivation of the shortest weighted processing time (SWPT) rule in which jobs are sequenced in order of nondecreasing processing time to weight ratios [Smith, 1956]. In addition to providing optimal solutions to single-machine problems, these orderings are used as priority rules for scheduling more complex systems.

A major theme in recent research has been the use of complexity theory to classify scheduling problems as polynomially solvable or NP-hard. Many fundamental results in this area are derived by Lenstra, Rinnooy Kan & Brucker [1977]. The NP-hardness of a problem suggests that it is impossible to find an optimal solution without the use of an essentially enumerative algorithm, for which computation times will increase exponentially with problem size. To obtain exact solutions of NP-hard scheduling problems, a branch-and-bound or dynamic programming algorithm is usually applied. In most cases where these algorithms have been successful in solving problems of reasonable size, problem-specific features are used to restrict the search.

In practice it may be acceptable to use a heuristic method to find an approximate solution for an NP-hard problem. There is clearly a trade-off between the computational investment in obtaining a solution and the quality of that solution. The performance of heuristic methods is often evaluated empirically, but it is sometimes possible to carry out a theoretical analysis of heuristic performance. Following the pioneering work of Graham [1966, 1969] on list

scheduling heuristics for parallel machines, there has been a significant interest in obtaining performance guarantees through worst-case analysis. An alternative approach is to use probabilistic analysis to predict the behavior for 'typical' problem instances.

There are, however, some classes of problems that have resisted attempts to design a satisfactory solution procedure: enumerative algorithms may be unable to solve problems with more than a handful of jobs, and the solutions generated by simple heuristic methods may be far from the optimum. Such problems can be tackled by local search methods. These methods have the advantage that they can be employed as 'black-box' techniques if no problem-specific knowledge is available. On the other hand, it is often possible to incorporate structural properties of a problem into the local search method to improve its performance.

In addition to reviewing the literature and presenting the results of computational tests, this chapter builds a framework for the application of local search methods to scheduling problems. Section 2 starts with a description of scheduling problems and then presents a classical representation scheme based on the physical environment and the performance criterion for the problem. The combinatorial nature of the various problem types is also identified; for example, many problems involve sequencing jobs or assigning jobs to machines. Section 3 discusses design features for local search algorithms including the representation of solutions and issues of feasibility. Section 4 outlines a range of possible local search algorithms for each of the broad classes of combinatorial structure that are encountered in scheduling problems. Specific applications of local search methods are reviewed in Sections 5, 6, and 7, within the general categories of single-machine, parallel-machine, and multi-stage problems. Section 8 compares the computational performance of different local search methods for four different types of scheduling problem. Section 9 makes some concluding remarks.

2 SCHEDULING MODELS

The machine scheduling problems that we consider can be described as follows. There are *m machines*, which are used to process *n jobs*. A *schedule* specifies, for each machine i and each job j, one or more time intervals throughout which processing is performed on j by i. A schedule is *feasible* if there is no overlapping of time intervals corresponding to the same job (so that a job cannot be processed by two machines at once), or of time intervals corresponding to the same machine (so that a machine cannot process two jobs at the same time), and also if it satisfies various requirements relating to the specific problem type. The problem type is specified by the machine environment, the job characteristics, and an optimality criterion.

2.1 Machine environment

Different configurations of machines are possible. In each case, however, all machines become available to process jobs at time zero.

A *single-stage* production system requires one operation for each job, whereas in *multi-stage* systems there are jobs that require operations on different machines. Single-stage systems involve either a single machine, or m machines operating in parallel. In the case of parallel machines, each machine has the same function. We consider three cases: *identical parallel machines* in which each processing time is independent of the machine performing the operation; *uniform parallel machines* in which the machines operate at different speeds but are otherwise identical; and *unrelated parallel machines* in which the processing time of an operation depends on the machine assignment.

There are three main types of *multi-stage* systems. All such systems that we consider contain m machines, each having a different function. In a *flow shop* with m stages, each job is processed on machines $1, \ldots, m$ in that order. In an *open shop* each job is also processed once on each machine, but the machine routing (that specifies the sequence of machines through which a job must pass) can differ between jobs and forms part of the decision process. In a *job shop* each job has a prescribed routing through the machines, and the routing may differ from job to job.

2.2 Job characteristics

The processing requirements of each job j are given: for the case of a single machine and identical parallel machines, p_j is the processing time; for uniform parallel machines, the processing time on machine i may be expressed as p_j/u_i, where u_i is the speed of machine i; for the case of unrelated parallel machines, a flow shop and an open shop, p_{ij} is the processing time on machine i; and for a job shop, p_{ij} denotes the processing time of the ith operation (which is not necessarily performed on machine i).

In addition to its processing requirements, a job is characterized by its availability for processing, any setup requirements on the machine, any dependence on other jobs, and whether interruptions in the processing of its operations are allowed. The availability of each job j may be affected by its *release date* r_j, which is the time that it becomes available for processing, or by its *deadline* $\bar{d}_j$, which specifies the time by which it must be completed.

Suppose that job j is sequenced immediately before job k on machine i. Then, after job j has completed processing on machine i, a *job setup time* t_{ijk} may be necessary before job k can start on i. If job k is scheduled first on machine i, the setup time required is t_{i0k}. Sometimes similar jobs share a setup. Specifically, suppose that jobs are partitioned into F families according to the similarity of their production requirements so that no setup on a machine is required between two consecutively sequenced jobs of the same family. The *family setup time* on machine i when a job of family g is immediately preceded by a job of family f is s_{ifg}, or s_{i0g} if there is no preceding job. If, for each k, we can write $t_{ijk} = t_{i0k} = t_{ik}$ for all jobs $j \neq k$, or if, for each g, $s_{ifg} = s_{i0g} = s_{ig}$ for all $f \neq g$ in the case of families, then the setup times on machine i are *sequence independent*.

Job dependence arises when there are *precedence constraints* on the jobs. If job j has precedence over job k, then k cannot start its processing until j is completed. Some scheduling models allow *preemption*: the processing of any operation may be interrupted and resumed at a later time. However, we restrict our attention to nonpreemptive scheduling.

2.3 Optimality criteria

For each job j, a *due date* d_j and a positive *weight* w_j may be specified. Given a schedule, we can compute for job j: the *completion time* C_j; the *lateness* $L_j = C_j - d_j$; the *earliness* $E_j = \max\{d_j - C_j, 0\}$; the *tardiness* $T_j = \max\{C_j - d_j, 0\}$; and the *unit penalty* $U_j = 1$ if $C_j > d_j$, $U_j = 0$ otherwise.

Some commonly used optimality criteria involve the minimization of: the maximum completion time $C_{\max} = \max_j C_j$; the maximum lateness $L_{\max} = \max_j L_j$; the total (weighted) completion time $\sum_j (w_j) C_j$; the total (weighted) tardiness $\sum_j (w_j) T_j$; the (weighted) number of late jobs $\sum_j (w_j) U_j$; or the total (weighted) earliness $\sum_j (w_j) E_j$; where each maximization and each summation is taken over all jobs j.

In some situations there is a setup cost, either in addition to or instead of a setup time. This is an example where it may be appropriate to adopt a composite objective which requires a weighted sum of two or more criteria to be minimized.

2.4 Three-field representation

It is convenient to adopt the representation scheme of Graham et al. [1979]. This is a three-field descriptor $\alpha|\beta|\gamma$ which indicates problem type: α represents the machine environment, β defines the job characteristics and γ is the optimality criterion.

Let $\circ$ denote the empty symbol. The first field takes the form $\alpha = \alpha_1 \alpha_2$, where α_1 and α_2 are interpreted as follows:

- $\alpha_1 \in \{\circ, P, Q, R, F, O, J\}$
  - $\alpha_1 = \circ$: a single machine
  - $\alpha_1 = P$: identical parallel machines
  - $\alpha_1 = Q$: uniform parallel machines
  - $\alpha_1 = R$: unrelated parallel machines
  - $\alpha_1 = F$: a flow shop
  - $\alpha_1 = O$: an open shop
  - $\alpha_1 = J$: a job shop
- $\alpha_2 \in \{\circ, m\}$
  - $\alpha_2 = \circ$: the number of machines is arbitrary
  - $\alpha_2 = m$: there are a fixed number of machines m

We note that for a single-machine problem $\alpha_1 = \circ$ and $\alpha_2 = 1$, whereas $\alpha_1 \neq \circ$ and $\alpha_2 \neq 1$ for other problem types.

The second field $\beta \subseteq \{\beta_1, \beta_2, \beta_3, \beta_4\}$ indicates job characteristics as follows:

- $\beta_1 \in \{\circ, r_j\}$
  - $\beta_1 = \circ$: no release dates are specified
  - $\beta_1 = r_j$: jobs have release dates
- $\beta_2 \in \{\circ, \bar{d}_j\}$
  - $\beta_2 = \circ$: no deadlines are specified
  - $\beta_2 = \bar{d}_j$: jobs have deadlines
- $\beta_3 \in \{\circ, t_{jk}, t_j, s_{fg}, s_f\}$
  - $\beta_3 = \circ$: there are no setup times
  - $\beta_3 = t_{jk}$: there are general job setup times
  - $\beta_3 = t_j$: there are sequence independent job setup times
  - $\beta_3 = s_{fg}$: there are general family setup times
  - $\beta_3 = s_f$: there are sequence independent family setup times
- $\beta_4 \in \{\circ, prec\}$
  - $\beta_4 = \circ$: no precedence constraints are specified
  - $\beta_4 = prec$: jobs have precedence constraints

Lastly, the third field defines the optimality criterion, which involves the minimization of

$$\gamma \in \{C_{\max}, L_{\max}, \sum (w_j) C_j, \sum (w_j) T_j, \sum (w_j) U_j, \sum (w_j) E_j\}.$$

Furthermore, as indicated in Section 2.3, it is sometimes appropriate to adopt a composite objective, one component of which may be a setup cost.

2.5 Some examples

To illustrate the three-field descriptor and to indicate the combinatorial nature of some scheduling problems, we present six examples.

$1 \| \sum w_j U_j$ is the problem of scheduling jobs on a single machine to minimize the weighted number of late jobs. An optimal solution can be assumed to have the property that the on-time jobs are sequenced first in nondecreasing order of their due dates, then the late jobs are sequenced in an arbitrary order after all the on-time jobs [Moore, 1968]. Thus, a *partition* of jobs into those that are on time and those that are late defines a solution.

$1 | s_f | \sum w_j C_j$ is the problem of scheduling families of jobs on a single machine to minimize the total weighted completion time, where a sequence independent setup time is necessary whenever the machine switches to processing jobs from a different family. Jobs within each family should be sequenced in nondecreasing order of p_j/w_j (SWPT order) [Monma & Potts, 1989]. Thus, a solution is obtained by *merging* the ordered lists of jobs for the different families.

$Pm | r_j | \sum C_j$ is the problem of scheduling jobs with release dates on a fixed number m of identical parallel machines to minimize the total completion time. After being assigned to machines, a sequence of jobs is required for each machine. Thus, a schedule is constructed using *combined assignment and sequencing.*

$R \| C_{\max}$ is the problem of scheduling jobs on an arbitrary number of unrelated parallel machines to minimize the maximum completion time. A solution is specified by an *assignment* of jobs to machines; the order in which jobs are processed on a machine is immaterial.

$F2 | prec | \sum w_j T_j$ is the problem of scheduling jobs with precedence constraints in a two-machine flow shop to minimize the total weighted tardiness. Since there exists an optimal schedule in which the same processing order is used on both machines [Conway, Maxwell & Miller, 1967], a schedule is obtained by *sequencing* the jobs.

$J \| C_{\max}$ is the problem of scheduling a job shop to minimize the maximum completion time. It is a *multisequencing* problem; implicit in a schedule is a sequence of operations on each of the machines.

3 APPLYING LOCAL SEARCH

3.1 Representation of solutions

For most applications of local search to scheduling problems, there is a natural way to represent the solutions. In a sequencing problem the *natural representation* of a solution is a permutation of the integers $1, \ldots, n$; for an assignment problem it is a list of the machines to which the respective jobs are assigned. In many cases there are, as we shall see, corresponding natural ways to define both a neighborhood structure and the operators required for a genetic algorithm. In this situation, a local search method can be applied as a 'black-box' technique, with the user required only to choose the algorithm parameters. If we wish to incorporate some problem-specific knowledge, this can sometimes be achieved by making a nonstandard choice of neighborhood structure, or by restricting the solutions that are allowed. Alternatively, a different representation for solutions of the problem may be more convenient.

One example of an alternative representation occurs for the problem $P \| \sum w_j T_j$ of scheduling identical parallel machines to minimize total weighted tardiness. It is clear that, in an optimal solution, the workload assigned to different machines should be balanced; solutions in which some job starts after another machine has completed all its processing can be discarded. By using a *list scheduling* method, a list of jobs in priority order can be used to define a schedule as follows. The first unscheduled job in the list is scheduled on the machine that first becomes idle; this process is repeated until all jobs are assigned to machines.

We can give a more formal description of these ideas by distinguishing between the set $\mathcal{S}$ of feasible (or restricted) solutions to the scheduling problem and some set $\mathcal{R}$ of representations upon which the local search will be performed. We use G to denote the map from $\mathcal{R}$ to $\mathcal{S}$ which corresponds to the method that is used to generate a solution from its representation. If there is just one way to represent any possible schedule, so that G is injective, it will make little difference whether we work with $\mathcal{R}$ or $\mathcal{S}$. In this case it is possible to translate the description of the algorithm from one set to the other without essential difference; so a neighbor-

hood structure in $\mathscr{R}$ implies a neighborhood structure in $\mathscr{S}$, and similarly, genetic crossover and mutation in $\mathscr{R}$ can be translated into corresponding operators in $\mathscr{S}$. On the other hand, if G is not injective, so that a single solution may have several representations, it will not be possible to translate the local search procedure into its equivalent in $\mathscr{S}$.

Returning to the problem $P \| \sum w_j T_j$, the natural representation is a list of processing orders, where each processing order defines the sequence of jobs assigned to the corresponding machine. The restricted set $\mathscr{S}$ of solutions is the set of lists of processing orders which produced a balanced schedule, as defined above. The set of all lists of job priority orders provides us with a *priority representation* $\mathscr{R}$, and G is the list scheduling method. In this case G is not injective, as a single solution may have several representations in $\mathscr{R}$.

A more complex way to use the idea of representation is to view a schedule as arising from the application of some rule or map H to data D. Here a representation of a schedule S is specified by a pair (D, H), with $H(D) = S$. Then the set $\mathscr{R}$ is given by some subset of $\mathscr{D} \times \mathscr{H}$, where $\mathscr{D}$ is a domain of possible data and $\mathscr{H}$ is a set of possible maps. We need $\mathscr{R}$ to be chosen in such a way that $H(D)$ is a feasible schedule for every $(D, H) \in \mathscr{R}$. Storer, Wu & Vaccari [1992] suggest that $\mathscr{H}$ is formed from a collection of heuristics appropriate to the problem at hand, and that $\mathscr{D}$ consists of different perturbations of the input data. It is natural to allow variations in just one of the two components of a representation, so we perform a local search either on $\mathscr{H}$ alone or on $\mathscr{D}$ alone.

If we perform a local search on a set of heuristics $\mathscr{H}$, the heuristics are effectively the different representations of the solutions. There are many different possible ways to define such a representation. For example, consider a single-machine sequencing problem where a variety of different 'basic' heuristics could be applied (such as scheduling in order of nondecreasing due dates or of nondecreasing processing times). Composite heuristics can then be constructed by allowing a different heuristic rule to be applied at each position in the sequence.

If a local search is performed on a set $\mathscr{D}$, which contains perturbations of the input data D, we must specify a fixed heuristic H. In a sequencing problem, for example, H might be the SWPT heuristic. This can be applied to input data with perturbed weights and processing times to generate a sequence of jobs. The sequence associated with a particular perturbation is then evaluated with respect to the original data D.

3.2 Neighborhood search

Neighborhood search algorithms include descent, simulated annealing, threshold accepting, and tabu search. The starting point for a neighborhood search method is the choice of neighborhood structure; this determines which pairs of solutions are regarded as adjacent to each other, and hence the moves that are allowed. Different neighborhoods have different properties, which may make them more or less suitable depending on the particular problem at hand. It is worth mentioning three in particular.

First, neighborhoods differ in *size*, i.e., in the number of neighbors for a single solution. Small neighborhoods are preferable from the point of view of the speed at which the local search proceeds. If the neighborhood is too small, however, the final solution may be of poor quality.

Second, neighborhoods differ in the *ease* with which new solution values can be computed. Neighborhood search techniques really come into their own when a move to an adjacent solution allows a rapid update of the objective function value, rather than a complete recalculation.

Third, the underlying *topology* of the neighborhood structure significantly affects the quality of solutions generated by a neighborhood search algorithm. We can think of the definition of a neighborhood as giving rise to an objective function *surface*. Specifying which pairs of solutions are neighbors implies a distance measure between solutions (the number of moves necessary to travel between them). If we could position the possible solution points in a plane so that the distances were correctly represented, then the local search procedure would move around on the surface generated by assigning objective function values as the vertical component at each solution point. A local optimum with respect to the neighborhood definition would then correspond to a local optimum in a classical continuous optimization sense. If the surface were very bumpy, there would be many local minima and a local search procedure might have trouble locating a good solution; on the other hand, if the surface were nicely behaved, a neighborhood search procedure could easily move downhill to find a very good solution. Among the different neighborhood structures for a particular problem, we prefer one for which the associated objective function surface is reasonably smooth. Unfortunately, however, it is difficult to define an easily computable measure of smoothness.

3.3 Genetic algorithms

The picture is more complicated for genetic algorithms than for neighborhood search. As with all local search methods, we first need to decide on how solutions are to be represented (usually in the form of a string). Secondly, a method is required to produce new solutions from two 'parent' solutions using some type of crossover operation. Although crossover is often regarded as a mechanical process that is performed on the string representation, a broader interpretation may be assumed. For example, a crossover operation might involve performing a sequence of moves in some suitably defined neighborhood on one of the parent solutions, where the characteristics of the other parent indicate which moves are candidates for consideration. After reversing the role of the two parent solutions, the process is repeated to produce another new solution. Thirdly, a suitable implementation of mutation is needed. Mutation can usually be thought of as a move to a random neighbor of the current solution, within some appropriately defined neighborhood. The process of generating new solutions using crossover and mutation applied to a suitable string is known as recombination.

A representation and a crossover method have to be selected together. There is often a choice between a simple representation which implies a complex crossover mechanism, and a complex representation for which a simple crossover method can be applied. The aim is to enable two parent solutions to combine to produce a new solution that has good features from both parents. Thus, crossover should try to preserve as many as possible of the elements of the parent solutions that are thought to be important in determining the quality of a solution. These are the so-called *building blocks* from which a new solution can be generated.

3.4 Neural networks

The natural approach when tackling a scheduling problem using a neural network is to adopt the methodology of Hopfield & Tank [1985]. Essentially, the problem is represented as one of constraint satisfaction.

In the Hopfield-Tank model, the outputs of the neurons, which are usually in the range zero to one, are used to define a solution. Thus, the problem should be represented by variables that take the value zero or one, so a zero-one programming formulation is useful. The 'system dynamics' of the network provide changes of state so that an energy function is decreased until it reaches a local minimum. An appropriate energy function includes the objective function for the scheduling problem, to which penalty costs corresponding to the constraints are added. The design of the network influences the success in obtaining a global minimum of the energy function.

Zero–one (or integer) programming formulations are available for most scheduling problems. Moreover, it is straightforward to convert these formulations into an energy function that can be used in the neural network. Thus, we shall not deal explicitly with these issues.

3.5 Feasibility

As pointed out in Section 2, the set of feasible schedules for a problem may be restricted by certain characteristics of the jobs, such as deadlines or precedence constraints. In other cases, we may know some structural properties of an optimal solution a priori, and it may be advantageous to treat this structural knowledge as an effective constraint on the schedules that we consider. In either circumstance, there are essentially four possible ways to proceed:

- *Exclusion.* Infeasible solutions are never considered, either because they are never generated or because they are always discarded. In neighborhood search it may be possible to employ some mechanism to ensure that only those neighbors that represent feasible solutions are generated. Alternatively, an infeasible neighbor can be discarded immediately it is generated. In a genetic algorithm, infeasible solutions are excluded by ensuring that the representation of schedules matches precisely those solutions we wish to consider.

- *Penalization.* Infeasible solutions are allowed but are penalized by adding a penalty cost to the objective function.
- *Repair.* After an infeasible solution has been generated, it is changed or repaired so that it becomes feasible. Thus, a method is required to find a feasible solution that retains the essential characteristics of the infeasible solution which has been generated.
- *Translation.* A repair mechanism can be used differently by regarding it as a translator; the original infeasible solution is left unaltered but is assigned an objective function value based on its feasible (translated) version. The local search is carried out on possibly infeasible solutions. In general different solutions may yield the same translated solution.

In neighborhood search, the simplest strategy to adopt is exclusion. It is implemented either by using some mechanism in the neighbor generation procedure to ensure that only feasible solutions are produced, or by discarding an infeasible solution immediately it is generated. For tightly constrained problems, however, to pass from one side of an infeasibility barrier to the other, so that another part of the solution space can be searched, may be difficult or impossible under exclusion. A penalty function approach may perform better. The repair and translation strategies should be used with caution because of the computational expense associated with applying the repair mechanism. Translation has the advantage that, like penalization, it allows an infeasibility barrier to be crossed.

For genetic algorithms, the exclusion strategy may be inappropriate because of the unavailability of a representation of solutions which avoids infeasibility. A penalty function approach is possible since a standard selection mechanism eliminates high-cost solutions. Alternatively, repair or translation strategies can be used. A possible advantage of translation is that the genetic information in the infeasible solution is left undisturbed, and could reemerge fruitfully in some later generation.

4 NEIGHBORHOOD STRUCTURES AND RECOMBINATION OPERATORS

Most scheduling problems fall into one of three types, according to their natural solution space representation. Some problems have solutions that are just a sequence of jobs; some problems have solutions that are assignments of jobs to machines, where a sequence of the assigned jobs may also be specified for each machine; and some problems have solutions that are multisequences – a set of sequences of operations, one for each machine. Our discussion also includes problems in which the natural representation is a partition of jobs into two subsets and the merging of ordered lists of jobs. Other natural representations of solutions do exist, but they occur less frequently. In this section, for each type of natural representation of solutions, we discuss alternative representations, give different neighborhoods for neighborhood search algorithms, and suggest various recombination operators for genetic algorithms.

4.1 Sequencing problems

There are many machine scheduling problems in which solutions are most naturally represented by stating the order, or sequence, in which the jobs are processed. Most of the single-machine problems that we discuss in this chapter are of this type.

Representation

The *natural representation* for sequencing problems is a permutation of the integers $1, \ldots, n$. However, there are other useful representations, three of which are given below.

In a *start time representation*, the start time of each job is specified. Applying local search operators to this representation is unlikely to produce feasible start times. Nevertheless, there is an obvious repair/translation mechanism: sequence the jobs in nondecreasing order of the start times resulting from the application of the local search operators.

If there are precedence constraints on the jobs, many sequences are infeasible. To overcome this drawback, we can use a *priority representation*, which is a priority order of jobs. A job becomes available when all of its predecessors are sequenced. We construct a feasible solution by selecting the first available unsequenced job in the priority order, and sequencing it next. Similar priority representations can be defined when there are release dates or deadlines; the essential difference is how availability of jobs is defined.

For scheduling problems of a sequencing nature, the crucial decisions may relate to whether one job is sequenced before or after another. This suggests an *ordered pair representation* that records, for each possible pair of jobs, which job comes before the other in the sequence. Thus, a sequence of six jobs (B, E, F, C, A, D) could be represented by a 15-element binary string as follows:

AB	AC	AD	AE	AF	BC	BD	BE	BF	CD	CE	CF	DE	DF	EF
0	0	1	0	0	1	1	1	1	1	0	0	0	0	1

Here the 0 against AB represents the fact that A does not appear before B in the sequence, whereas the 1 against BC indicates that B is sequenced before C. After applying local search operators to such a binary string, there is little likelihood that the result is a feasible solution. Nevertheless, this is not a problem if, as suggested in Section 3.5, a repair or translation operator is applied to the string. Suppose that we construct a directed graph G from the string in which vertices correspond to the jobs $1, \ldots, n$, and for each pair of jobs j and k there is an arc (j, k), or (k, j), according to whether or not the string specifies that j precedes k. The 'best' repair/translation operator would find an acyclic subgraph of G containing a maximum number of the arcs of G, which defines a sequence. However, since this *acyclic subgraph problem* is NP-hard – feedback arc set is NP-hard [Karp, 1972] – it is more appropriate to apply a constructive heuristic. For example, a greedy heuristic may be used: select a job for the first unfilled position

in the sequence such that this decision causes as few arcs as possible in G to be deleted.

Neighborhoods

Consider first the natural representation for a sequencing problem. We give four possible neighborhoods below; each is illustrated by considering a typical neighbor of the sequence (A, B, C, D, E, F, G, H) in a problem where there are eight jobs labeled $A, \ldots, H$.

- *Transpose.* Swap two adjacent jobs. Thus, $(A, B, \mathbf{D}, \mathbf{C}, E, F, G, H)$ is a neighbor.
- *Insert.* Remove a job from one position in the sequence and insert it at another position (either before or after the original position). Thus, $(A, \mathbf{E}, B, C, D, F, G, H)$ and $(A, B, C, D, F, G, \mathbf{E}, H)$ are both neighbors.
- *Swap.* Swap two jobs that may not be adjacent. Thus, $(A, \mathbf{F}, C, D, E, \mathbf{B}, G, H)$ is a neighbor.
- *Block insert.* Move a subsequence of jobs from one position in the sequence and insert it at another position. Thus, $(A, \mathbf{D}, \mathbf{E}, B, C, F, G, H)$ is a neighbor.

This terminology is not universally adopted: *shift* is sometimes used instead of *insert*, and *interchange* instead of *swap*. Moreover, there are other possible more complex neighborhoods. For example, by analogy with the neighborhood structures that are frequently used for the traveling salesman problem, we can define *k-opt* neighborhoods that move (and possibly reverse) more than one subsequence in the schedule.

As indicated in Section 3, size is an important consideration when choosing between neighborhoods. In calculating the neighborhood sizes, we must avoid double counting; for example, starting from (A, B, C, D, E, F, G, H), inserting job B after job C has the same result as inserting job C before job B. With n jobs to be sequenced, the neighborhood sizes are $n-1$ for transpose; $(n-1)^2$ for insert; $n(n-1)/2$ for swap; and $n(n+1)(n-1)/6$ for block insert. The neighborhood with the smallest size is not necessarily preferred because the topology of the solution space also influences our choice.

Another factor that affects the choice of neighborhood is the computational effort required to evaluate a neighbor. For many single-machine problems, a rapid update of the objective function is possible. Consider, as an example, the problem $1 \| \sum T_j$ of minimizing total tardiness on a single machine. Suppose that we start with the sequence $(1, \ldots, n)$. Then, to calculate the effect of inserting job j in position k $(k > j)$, we note that the tardiness T_l of job l, for $l = j+1, \ldots, k$, is reduced by $\min\{p_j, T_l\}$, whereas the lateness for job j is increased by $\sum_{l=j+1}^{k} p_l$. Since the tardiness of the other jobs is not altered, the change in total tardiness is easily computed. Similar arguments are possible for the other neighborhoods we have given.

For the priority representation mentioned above, exactly the same neighborhoods can be used. For the start time and ordered pair representations, it is more natural to consider changes to one or more elements of the representation, rather

than moving elements around. Thus, for example, a neighborhood for the start time representation can be obtained by allowing a single start time to change, with some limit imposed on the maximum increase or decrease. A natural neighborhood for the ordered pair representation is obtained by allowing up to two of the binary elements in the string to change (from 0 to 1, or vice versa).

Recombination

We first consider a natural representation of solutions in which each string is a permutation of the integers $1,\ldots,n$. For sequencing problems there are many methods of combining two solutions to produce two new solutions. We describe three of them below. Each of our descriptions relates to crossover operations that are defined by two randomly selected crossover points. For corresponding one-point crossovers, a single crossover point is randomly chosen and, by implication, a second point occurs at the beginning or the end of the sequence.

In a *partially matched crossover*, two positions are picked at random and the sections or subsequences of the two sequences between these positions are then interchanged element by element. This is much easier to describe with an example. Suppose we start with sequences

$$\pi_1 = A\ B\ C \mid D\ E\ F\ G \mid H\ I\ J$$
$$\pi_2 = C\ H\ G \mid A\ I\ D\ B \mid J\ F\ E$$

The two strings change places between the crossover points, and outside the crossover points the reverse changes are made in order to maintain valid sequences. We obtain

$$\pi_1' = F\ G\ C \mid A\ I\ D\ B \mid H\ E\ J$$
$$\pi_2' = C\ H\ B \mid D\ E\ F\ G \mid J\ A\ I$$

Here A is swapped with D and then D with F, to give the first element in π_1'. B is swapped with G to give the second element, and so on.

The *insertion crossover* is a version of the crossover operation which has been used successfully by Mühlenbein and coworkers for the traveling salesman problem [e.g., Mühlenbein, Gorges-Schleuter & Krämer, 1988]. Two positions are picked at random and the sections or subsequences between these points are then inserted into the other sequence at one of the point where an element in the subsequence occurs (the choice being made randomly). Elements in the inserted subsequence are removed from the other places where they occur, and the gaps closed up. If we take π_1 and π_2 as above, this crossover might produce

$$\pi_1' = \mid A\ I\ D\ B \mid C\ E\ F\ G\ H\ J$$
$$\pi_2' = C\ H\ A\ I \mid D\ E\ F\ G \mid B\ J$$

We have inserted the substring from π_2 in place of A in π_1, and the substring from π_1 in place of D in π_2. The end result is that a subsequence from one parent is

inserted into the other, with the remaining elements in the sequence occurring in the same order as before.

A *reorder crossover* picks two positions at random, as before, and then the subsequences between them are reordered to match the order of the elements in the other sequence. If we take π_1 and π_2 as above, this crossover might produce

$$\pi_1' = A\ B\ C \mid G\ D\ F\ E \mid H\ I\ J$$
$$\pi_2' = C\ H\ G \mid A\ B\ D\ I \mid J\ F\ E$$

These three types of crossover can also be applied to the priority representation in which the priority order is a sequence of jobs. For start time and ordered pair representations, a standard crossover can be used before applying the necessary repair/translation operators.

4.2 Assignment and partitioning problems

There are several scheduling problems whose solution can be viewed as an assignment of the jobs to machines. One example given in Section 2 is the problem $R \| C_{\max}$, which requires jobs to be assigned to unrelated parallel machines to minimize the maximum completion time. Another example involving assignment is $P \| L_{\max}$, in which jobs are to be scheduled on identical parallel machines to minimize the maximum lateness; once an assignment of jobs to machines is decided, the optimal schedule on each machine can be obtained by sequencing jobs in EDD order (nondecreasing order of due dates) [Jackson, 1955].

Some single-machine scheduling problems require jobs to be partitioned into two sets. An example is $1 \| \sum w_j U_j$, in which jobs are to be scheduled on a single machine to minimize the weighted number of late jobs. A partition of jobs into those which are late and those which are on time is sufficient to determine a schedule. The partition is constrained because not all subsets of jobs can be scheduled on time. Another example of a partitioning problem is $1|d_j = d|\sum (w_j E_j + w_j' T_j)$, in which jobs with a common due date d are to be scheduled on a single machine to minimize the total weighted earliness plus total weighted tardiness. Assume that $d \geqslant \sum_j p_j$. Baker & Scudder [1990] generalize the properties of an optimal schedule derived by Kanet [1981] for unit weights. Specifically, there is no idle time between jobs, and one job completes at time d. Furthermore, jobs that are completed no later than time d are sequenced in order of nonincreasing p_j/w_j (longest weighted processing time or LWPT order) and jobs completed after time d are sequenced in order of nondecreasing p_j/w_j' (SWPT order). Thus, a solution is specified by a partition of the jobs into those that the completed no later than time d and those that are completed after time d.

A partitioning problem is obviously a special case of an assignment problem, and it is convenient to treat the two types of problems together.

Representation

Suppose that we have a problem in which n jobs are to be assigned to m machines. The *natural representation* for an assignment problem is a set of m lists, where each list contains the jobs assigned to one particular machine. Equivalently, this representation can be a single list containing the respective machine assignments of the n jobs.

As in the case of sequencing problems, we can also adopt a *priority representation.* To obtain an assignment of jobs to machines from the priority order, *list scheduling* is applied. More precisely, whenever a machine becomes idle, the next job on the priority list is scheduled on that machine. Since the priority representation is a sequence of jobs, the neighborhoods and recombination operators of Section 4.1 are applicable.

Neighborhoods

For the natural representation of solutions to assignment problems, we give three possible neighborhoods:

- *Reassign.* Remove a job from one machine and reassign it to another.
- *Swap.* Swap two jobs from different machines by reversing their machine assignments.
- *2-Reassign.* Remove either one or two jobs and reassign them to different machines.

More generally, we could define a *k-reassign* neighborhood which allows the reassignment of up to k jobs. Neighborhood sizes are $n(m-1)$ for reassign and $n(m-1)[1+(n-1)(m-1)/2]$ for 2-reassign. For swap, the maximum possible neighborhood size occurs when jobs are evenly divided between the machines. Thus, an upper bound on the neighborhood size is $\lceil n/m \rceil^2 m(m-1)/2$.

It may be preferable to reduce the neighborhood size using an exclusion mechanism. We define a job to be *critical* if any small delay to its start time causes an increase in the objective function value. As an example, for $R \| C_{\max}$, jobs on a most heavily loaded machine are critical for the maximum completion time objective. To obtain an improved solution, some critical job must be assigned to a different machine. Thus, we define the *critical reassign*, *critical swap*, and *critical k-reassign* neighborhoods, to be subsets of the original neighborhoods in which at least one critical job is moved to another machine.

Neighbors for assignment problems can often be evaluated quickly if appropriate values for the current solution are stored. For example, consider the problem $P \| \sum w_j C_j$ of scheduling jobs on identical parallel machines to minimize the total weighted completion time. Given an assignment of jobs to machines, the jobs on each machine are sequenced in SWPT order. Suppose that jobs $1, \ldots, k_1$ are sequenced in this order on some machine i_1, and jobs $k_2, \ldots, n$ are sequenced in this order on another machine i_2, and we wish to calculate the effect of removing job j from machine i_1 (where $j \leqslant k_1$) and reassigning it on machine i_2. Assume that,

in the new SWPT ordering on machine i_2, job j is sequenced immediately after job l (where $k_2 \leqslant l \leqslant n$). The total weighted completion time of jobs $j+1,\ldots,k_1$ on machine i_1 decreases by $p_j \sum_{h=j+1}^{k_1} w_h$, the total weighted completion time of jobs $l+1,\ldots,n$ on machine i_2 increases by $p_j \sum_{h=l+1}^{n} w_h$, and the weighted completion time of job j decreases by $w_j(\sum_{h=1}^{j-1} p_h - \sum_{h=k_2}^{l} p_h)$ (which may be negative). Since the weighted completion time of other jobs is unaltered, the change in the total weighted completion time is easily computed if, for the current schedule, partial, sums of weights and processing times on each machine are available. A similar efficient recompuation of the total weighted completion time is also possible for the swap and 2-reassign neighborhoods.

Recombination

Consider the natural representation of solutions. The application of genetic algorithms in this case is particularly easy, since a string representation is immediately available by listing for each job the machine to which it is assigned. Then crossover can be carried out using a standard one- or two-point method. In contrast to sequencing problems where standard crossover operations fail to yield a sequence, any list of machine numbers defines a possible assignment. It is worth noting, however, that the operation of the genetic algorithm will be affected by the order of the elements in the string. The jobs that are placed close together on the string are more likely to retain their relationship with each other under the standard crossover operation. Unfortunately, in most such problems, it is very hard to see what might be an appropriate ordering of the jobs. One option is to use a different form of crossover operation. Using the *uniform crossover*, two solutions can be mated by choosing randomly at each point in the string which parent should provide the element for the child string. The other child is obtained by making the opposite choice at each point. This will make the order of the elements in the string unimportant. If we think of genetic algorithms as working with building blocks, out of which good solutions can be put together, then the building blocks will be just the individual job assignments.

4.3 Combined assignment and sequencing problems

There are several problems in which an assignment of jobs to machines and a sequencing of the jobs on the machines are both necessary (e.g., the problem $P|r_j|\sum C_j$ of scheduling jobs with release dates on m identical parallel machines to minimize the total completion time). Since they contain elements of pure sequencing and assignment problems, generalizations of some of the approaches discussed above are appropriate.

Representation

The *natural representation* of solutions is a set of m ordered lists, where each list contains the sequence of jobs assigned to one particular machine.

Kanet & Sridharan [1991] suggest a *start time representation* which specifies, for each job, a vector containing its machine assignment and start time. Local search can be applied to this list of vectors in much the same way as for the start time representation of a sequencing problem, except that changes in job assignments are possible as well as changes in start times. As before, a repair/translation procedure is necessary for the start times obtained from the local search operators: for each machine the assigned jobs are sequenced in nondecreasing order of these start times.

Since the *priority representation* for pure assignment problems uses list scheduling to create a schedule, it also provides a valid representation for our combined assignment and sequencing problems. The neighborhoods and recombination operators of Section 4.1 can be applied to the priority sequence.

Neighborhoods

For a natural representation of solutions, the neighborhoods described for the pure sequencing problems can be generalized. Thus, *insert* removes a job from its current position on some machine and inserts it in a different position, either on the same machine or on a different machine. Also, jobs on the same or on different machines can be interchanged using *swap*.

Recombination

It is difficult to specify a method for recombination when the natural representation is adopted. For the start time representation, however, it is straightforward to apply crossover to lists of vectors. As pointed out in Section 4.2, different job orderings give different lists, and it is difficult to select a list that is suitable if standard one- or two-point crossover is used. However, uniform crossover avoids the need to choose a job ordering.

4.4 Multisequencing problems

Most multi-stage scheduling problems, such as a flow shop and job shop, require operations to be sequenced on several machines, so we call them multisequencing problems.

Representation

We first describe a *natural representation*. Flow shop and job shop schedules are defined by a set of sequences, one for each machine. For the open shop, in addition to a sequence for each machine, the solution must also specify a machine routing for each job. Due to the interrelationships between the different sequences, the obvious generalizations of the approaches outlined in Section 4.1 are likely to be ineffective. For example, suppose that the insert neighborhood is used for some flow shop problem in which different sequences of jobs are allowed on

different machines. If a job is moved from one of the initial positions on a machine to one of the final positions, then the resulting schedule is likely to be of poor quality unless corresponding adjustments are made to the sequences on other machines. Now consider a job shop problem. If job j has to be processed first by machine 1 and then by machine 2, and job k is processed by these two machines in the reverse order, we cannot have a schedule in which k comes before j on machine 1 and k comes after j on machine 2. A simple-minded approach may therefore prove infeasible. Similar disadvantages of ineffectiveness and infeasibility also occur for the open shop.

A *priority representation* helps to counter these drawback. For the flow shop and job shop a priority order of operations is given for each machine. An operation becomes available when the previous operation on the same job is completed. When a machine becomes idle, an available operation with highest priority is started (or if there is none, the first operation to become available is scheduled next, ties being broken by the priority rule). This always creates a nondelay schedule. Although there may be no optimal nondelay schedule, this approach can still be effective in generating near-optimal solutions. Furthermore, this potential loss of optimality can be overcome by modifying the heuristic to generate active rather than nondelay schedules (an active schedule is one in which no operation can be processed earlier without delaying the start of another operation). For the open shop we can use a similar representation. Suppose that we specify for each machine a priority order for the jobs, and for each job a sequence of machines that defines its route. A schedule can then be constructed using the same method as for a job shop.

Neighborhoods

The priority representation rather than the natural representation of solutions is generally recommended for the reasons indicated above. Using the priority representation of solutions for the flow shop or job shop, it is straightforward to apply neighborhood search. Since the priority order on each machine is a sequence, it is possible to use one of the neighborhoods given in Section 4.1 for pure sequencing problems, such as insert or swap. Similarly, for the open shop, a neighborhood is defined by allowing a change in either one of the priority orders or in one of the job route sequences.

It is also possible to use a natural representation for the flow shop or job shop, but using an exclusion mechanism of the type introduced in Section 4.2, as proposed by Van Laarhoven, Aarts & Lenstra [1992]. Consider the minimization of the maximum completion time. As in Section 4.2, we define an operation to be *critical* if any small delay to its start time causes an increase in the objective function value. Furthermore, a maximal sequence of critical operations is called a *critical path*, and a corresponding maximal subsequence of critical operations on the same machine is called a *block*. To obtain an improved solution, some critical operation must start earlier by a suitable reordering of the operations in a block. If the transpose neighborhood is used for each of the sequences, a neighbor is considered only if two operations in the same block are swapped;

other neighbors are excluded. We refer to this neighborhood as *critical transpose*. It is shown by Van Laarhoven, Aarts & Lenstra that any solution obtained by critical transpose is feasible. A further restriction of this neighborhood is motivated by the observation that no improvement in the solution results by a reordering of the operations of a block if the first operation still precedes the others in the block and the last operation also remains at the end. The *critical end transpose* neighborhood is a restricted version of the critical transpose neighborhood in which the first or last operation of a block is transposed. Similarly, we define the *critical end insert* neighborhood as a restricted version of insert in which an operation is removed from a block and placed immediately before the first or immediately after the last operation in the block. For other optimality criteria, this type of exclusion approach is likely to be less effective because the number of critical operations is larger.

Under a natural representation, a *machine reschedule* neighborhood can be adopted for flow shop and job shop problems. In this neighborhood a machine is selected for rescheduling, but the processing order on the other machines remains fixed. A solution (either optimal or heuristic) of the resulting single-machine subproblem provides the new schedule. For the open shop, a similar approach can be used to reschedule a machine or to reroute a job through the machines. A more general *k-machine reschedule* neighborhood allows the rescheduling of up to k machines; this can be achieved by considering a k-machine subproblem, or by some sequence of k single-machine subproblems.

Recombination

For flow shop, job shop, and open shop problems, the priority representation can be used for genetic algorithms. In each case a solution is represented by a set of sequences and one of the methods of Section 4.1 can be used.

4.5 Merging ordered lists of jobs

Sometimes a problem with a more complex solution structure arises because we can deduce some properties of an optimal solution, and it is advantageous to restrict the search to solutions which have these required characteristics. An example of this phenomenon occurs for the problem $1|s_f|\sum w_jC_j$ in which families of jobs with associated sequence-independent setup times are to be scheduled on a single machine to minimize the total weighted completion time. This can be viewed as a sequencing problem. Since jobs within each family must be sequenced in a SWPT order [Monma & Potts, 1989], it can also be regarded as a problem of merging ordered lists of jobs, where each ordered list contains all the jobs in one family.

Representation

Since we know the order in which the jobs within a single family are to be processed, the *natural representation* of a schedule is obtained by simply indi-

cating the family of the job at each position in the schedule. Thus, we can represent a schedule for a problem with five jobs in each of three families by a string of the form $(A, C, C, A, A, A, B, B, B, C, C, C, B, A, B)$, where A, B, and C are the families.

It is sometimes the case that we can deduce further properties of an optimal schedule. In particular, if we decide on how each family is split up into batches, where a batch contains jobs that are to be scheduled contiguously, then we may determine the sequence of these batches. For example, in the problem $1|s_f|\sum w_jC_j$ there is a generalized SWPT rule which applies to complete batches as well as to individual jobs. This allows an alternative *batch-based representation*, which is obtained by using a binary string to mark the positions in each family at which a new batch begins. The schedule above is described by three binary strings $(1, 1, 0, 0, 1), (1, 0, 0, 1, 1), (1, 0, 1, 0, 0)$, where each string corresponds to a single family. If the jobs of family A are numbered $1, \ldots, 5$, then the first string $(1, 1, 0, 0, 1)$ indicates that the jobs of A are split into three batches that contain job 1, jobs 2 to 4, and job 5, respectively. Each of the binary strings must start with a 1, so we could omit the first element in each. Note also that the same schedule could arise from more than one set of binary strings; although we may choose to start a new batch at a particular position among the jobs from one family, the mechanism for sequencing the batches could schedule two batches of the same family consecutively.

Neighborhoods

Using the natural representation, it is necessary to retain the same number of elements, of each type in any changed string. This makes it appropriate to use the same type of neighborhoods as for the natural representation of sequencing problems (transpose, insert, swap, etc.). For this approach, some of the possible moves will not alter the resulting schedule (e.g., swapping two jobs of the same family). For each of the three neighborhoods transpose, insert, and swap, it is straightforward to specify the neighbors that will yield a different schedule and then to restrict attention to these neighbors.

One class of neighborhoods in the batch-based representation is defined by changing one or more elements in the solution representation (0 to 1, or vice versa). Once we have taken account of the fixed 1 at the start of each family's binary string, all other binary strings represent feasible schedules.

Recombination

If we use a natural representation, the following methods of performing a crossover operation are possible. First, we could use a version of one of the crossovers used for a pure sequencing problem. However, this approach effectively ignores the special structure of the merged strings problem. An alternative is to use a standard one- or two-point crossover operation followed by a translation/repair mechanism to obtain the correct number of elements from each

family. It is more satisfactory, however, to use a batch-based representation. In this case we can concatenate the strings for different families into a single string and then use a standard crossover operation.

5 SINGLE-MACHINE PROBLEMS

In this section we indicate how local search can be applied to a variety of single-machine scheduling problems. We exclude from our discussion problems of minimizing the maximum completion time; most problems of this type are polynomially solvable, so local search is inappropriate. For the other optimality criteria, we briefly review the main results and, where appropriate, provide a guide to problem 'hardness' by indicating the size of problem for which the best currently available (enumerative) algorithm can generate an optimal solution.

Since many single-machine scheduling problems require jobs to be sequenced, the guidelines given in Section 4.1 are especially useful.

5.1 Maximum lateness

The problem $1 \| L_{\max}$ is solved in $O(n \log n)$ time by Jackson's earliest due date (EDD) rule [Jackson, 1955]: jobs are sequenced in order of nondecreasing due dates. When there are precedence constraints, Lawler [1973] shows there exists an optimal schedule in which a job that has the largest due date among those with no successors is processed last. Repeated application of this result yields an optimal schedule for $1|prec|L_{\max}$ in $O(n^2)$ time. Although Lenstra, Rinnooy Kan & Brucker [1977] show that $1|r_j|L_{\max}$ is strongly NP-hard, branch-and-bound algorithms are able to solve large instances. For example, Carlier [1982] solves 10000-job problems using an ingenious branching rule and lower bounds obtained by allowing preemption. Monma & Potts [1989] show that $1|s_f|L_{\max}$ is solvable by dynamic programming in $O(F^2n^{2F})$ time (where F is the number of families), and Bruno & Downey [1978] prove NP-hardness in the ordinary sense for arbitrary F.

We now discuss the design of local search algorithms for the problem $1|r_j|L_{\max}$. The problem is of special interest since various algorithms for the job shop problem $J \| C_{\max}$ are based on solving a subproblem which is equivalent to $1|r_j|L_{\max}$. The search for an optimal solution to $1|r_j|L_{\max}$ can be restricted to *active* schedules, in which no job can be scheduled earlier without delaying the start of another job. Suppose first that a natural representation of solutions as a sequence is adopted, so that there is a corresponding schedule in which each job is processed as early as possible subject to its release date and the machine availability. One of the approaches of Section 4.1 can be applied, and a repair/translation procedure used if a nonactive schedule is generated. An alternative and possibly preferable approach is to use a priority representation. We construct an active schedule by processing next the first unscheduled job in the priority order for which its start time is strictly smaller than the earliest possible

completion time among all jobs that can be processed next. The methods of Section 4.1 can be applied to this priority order.

For $1|s_f|L_{\max}$ Monma & Potts [1989] show that jobs within each family are sequenced in EDD order. Thus, the problem requires the merging of ordered lists of jobs, where each list contains all jobs of a family in EDD order. Moreover, by associating a due date with each batch, the EDD rule can be applied to sequence the batches. Thus, the approaches of Section 4.5 that use a batch-based representation can be applied.

5.2 Total weighted completion time

The basic problem $1\|\sum w_jC_j$ is solved in $O(n \log n)$ time by Smith's shortest weighted processing time (SWPT) rule [Smith, 1956]: jobs are sequenced in order of nondecreasing ratios p_j/w_j. In the case that jobs have unit weights and have deadlines, Smith generalizes the SPT rule (the shortest processing time rule, which is a special case of the SWPT rule) by showing that, among jobs j for which $\bar{d}_j \geqslant \sum_{k=1}^{n} p_k$, a job with the largest processing time is processed last. Repeated application of this result yields an optimal schedule for $1|\bar{d}_j|\sum C_j$ in $O(n \log n)$ time. Most other extensions of the basic model are strongly NP-hard, including $1|r_j|\sum C_j$ [Lenstra, Rinnooy Kan & Brucker, 1977], $1|prec|\sum C_j$ [Lawler, 1978; Lenstra & Rinnooy Kan, 1978] and $1|\bar{d}_j|\sum w_jC_j$ [Lenstra, Rinnooy Kan & Brucker, 1977]. Although Ahn & Hyun [1990] and Ghosh [1994] show that $1|s_f|\sum C_j$ and $1|s_f|\sum w_jC_j$ are solvable by dynamic programming in $O(F^2n^F)$ time, the complexity of these problems is open when F is arbitrary.

Branch-and-bound algorithms proposed by Chu [1992] and Belouadah, Posner & Potts [1992] for $1|r_j|\sum C_j$ and $1|r_j|\sum w_jC_j$ are capable of solving instances with up to 100 and 40 jobs, respectively. Both algorithms rely heavily on dominance rules to restrict the search. The design principles of local search algorithms for $1|r_j|L_{\max}$ given in Section 5.1 can also be used for $1|r_j|\sum (w_j)C_j$. Thus, a natural representation of solutions can be used with a repair/translation mechanism applied to generate active schedules, or a priority representation can be adopted and the active schedule generation procedure of Section 5.1 used to create a schedule from the representation.

For $1|\bar{d}_j|\sum w_jC_j$ the branch-and-bound algorithms of Posner [1985] and Potts & Van Wassenhove [1983] solve instances with up to 40 jobs. Suppose that a natural representation of solutions is adopted. Infeasible sequences can discarded in neighborhood search. It is also possible to enforce the deadline constraints using a penalty function approach. A priority representation may alternatively be adopted. To construct a feasible sequence from a priority order, among jobs j for which $\bar{d}_j \geqslant \sum_{k=1}^{n} p_k$, the job that is nearest to the end of the priority order is processed last. Repeated application of this rule yields the required feasible sequence.

We now consider the problem $1|prec|\sum w_jC_j$. Computational results obtained by Potts [1985a] with his Lagrangean-based branch-and-bound algorithm show that instances with 100 jobs can be solved. Potts also proposes a descent

algorithm, which uses a block insert neighborhood on the natural representation of solutions and in which infeasible solutions are discarded. However, other neighborhood search algorithms can be designed. For example, the precedence constraints can be enforced using a penalty function approach. Another option is to use an ordered pair representation: by applying a suitable repair/translation mechanism, the resulting solution is guaranteed to be feasible (the greedy heuristic of Section 4.1 for the acyclic subgraph problem can be modified to ensure that all precedence constraints are satisfied).

For $1|s_f|\sum w_jC_j$ Mason & Anderson [1991] derive a branch-and-bound algorithm which relies mainly on dominance rules to restrict the search. Their computational results for $1|s_f|\sum C_j$ indicate that instances with up 30 jobs can be solved. Recall from Section 4.5 that the problem can be regarded as one of merging lists, where each list contains the jobs of a family in SWPT order. Adopting a natural representation of solutions, Ahn & Hyun [1990] propose a descent algorithm for $1|s_f|\sum C_j$ which uses the block insert neighborhood. By restricting the choice of blocks, it is possible to ensure that jobs within each family are sequenced in SWPT order. Mason [1992] proposes a genetic algorithm which uses the batch-based representation. Crauwels, Potts & Van Wassenhove [1997] design various neighborhood search algorithms. In multistart descent, simulated annealing, and threshold accepting, they use the block insert neighborhood of Ahn & Hyun, and the two latter methods use temperatures and threshold values that follow a periodic pattern. Their tabu search algorithm uses the restricted insert neighborhood, and a tabu list of length 7 stores the job that is inserted and prevents it from moving again. In a computational comparison of Mason's algorithm with these neighborhood search algorithms, Crauwels, Potts & Van Wassenhove find that all five methods generate solutions of high quality. Based on solution quality and computation time, tabu search is preferred to the other methods. These results are presented in Section 8.1.

Herrmann & Lee [1995] propose a genetic algorithm for the problem $1|\bar{d}_j, s_{fg}|\sum C_j$, which is shown by Bruno & Downey [1978] to the NP-hard. They use a representation which consists of binary encoding of perturbations of the original deadlines. To obtain a solution from the representation, a backward scheduling heuristic is used which aims to minimize the time spent on setups and to process the longer jobs as late as possible. The resulting schedule is not guaranteed to be feasible with respect to the original (or the perturbed) deadlines. Thus, a penalty function approach is used to drive the solution towards feasibility.

Laguna, Barnes & Glover [1991] propose tabu search algorithms for $1\|\sum w_jC_j + \sum c_{jk}$, in which there are general job setup costs and the objective is to minimize the total weighted completion time plus the total setup cost. They adopt a natural representation of solutions as a sequence, and compare a restricted version of the swap and insert neighborhoods which prohibits moves that change the position of a job by more than $n/2$. By storing partial sums of weights and processing times for the current sequence, candidate moves can be evaluated in constant time. For a swap move that interchanges the jobs in positions

j and k, where $j < k$, the tabu list entry prevents the first of these jobs from occupying the first j positions. After an insert move, the job that is removed and inserted in another position is stored on the tabu list and prevented from moving again. Computational results with tabu lists of length 7 indicate that the insert neighborhoods yields better quality solutions than the swap neighborhood. However, a slight improvement is observed, especially for larger problems, if a combined swap and interchange neighborhood is adopted. This approach is extended by Laguna, Barnes & Glover [1993] to the problem $1|t_{jk}|\sum w_jC_j + \sum c_{jk}$, in which there are job setup times and costs. Glover & Laguna [1991] and Laguna & Glover [1993] use 'target analysis' to develop an improved algorithm for $1\|\sum w_jC_j + \sum c_{jk}$. Target analysis is a technique which 'learns' heuristic rules that are appropriate for a class of problems. In this case, whenever the tabu search algorithm is forced to make a nonimproving move, an attempt is made to diversify the search by penalizing neighbors according to the frequency with which, in previous iterations, the same job pair has been swapped or the particular insert job is placed just before the same job. Results indicate that this modified tabu search algorithm is superior to the original version.

Arizono, Yamamoto & Ohta [1992] propose a neural network approach for $1|t_{jk}|\sum C_j$ (although they work with the equivalent problem $1|t_{jk}, \bar{d}_j = \bar{d}|\sum E_j$, in which jobs have a common deadline). Their model uses the output of a neuron to define whether or not a job occupies a particular position in the sequence.

5.3 Total weighted tardiness

For many years the complexity of $1\|\sum T_j$ remained open. In a recent paper, however, Du & Leung [1990] show that the problem is NP-hard in the ordinary sense. Lawler [1977] shows that it is pseudopolynomially solvable by deriving an $O(n^4\sum_j p_j)$ algorithm, which is based on decomposition. More precisely, a job with the largest processing time partitions the problem into two subproblems: jobs with the smallest due dates are sequenced before this partition job, and the remaining jobs are sequenced after it. A search is performed for the optimal position of the partition job. Using a dynamic programming algorithm of Schrage & Baker [1978] to solve subproblems, Potts & Van Wassenhove [1982] develop this decomposition approach to solve instances with up to 100 jobs.

Lawler [1977] and Lenstra, Rinnooy Kan & Brucker [1977] show that $1\|\sum w_jT_j$ is strongly NP-hard. Successful enumerative algorithms rely heavily on elimination criteria of the type derived by Emmons [1969] and Rinnooy Kan, Lageweg & Lenstra [1975], which give conditions under which certain jobs can be assumed to precede others in the search for an optimal schedule. The branch-and-bound algorithm of Potts & Van Wassenhove [1985], which uses a relatively weak but quickly computed lower bound, can solve problem instances with up to 40 jobs. A survey of algorithms is given by Abdul-Razaq, Potts & Van Wassenhove [1990].

There are various descent and simulated annealing algorithms which use natural representations of solutions as sequences and employ one of the neigh-

borhoods of Section 4.1. The algorithm of Wilkerson & Irwin [1971] for $1 \| \sum T_j$ resembles a descent algorithm that applies the transpose neighborhood to the EDD sequence. Chang, Matsuo & Tang [1990] also consider descent algorithms for this problem. They show that the ratio of total tardiness given by a descent algorithm to the minimum total tardiness (for instances in which the minimum total tardiness is positive) can be arbitrarily large if the transpose or swap neighborhood is used, although the ratio is finite for the insert neighborhood.

Matsuo, Suh & Sullivan [1987] apply simulated annealing to $1 \| \sum w_j T_j$. In their algorithm the probability of accepting a worse solution is independent of the objective function values. Based on computational results with the transpose neighborhood, they find that a systematic search of the neighborhood outperforms a random search and that the use of a good initial sequence accelerates the process of finding a near-optimal solution. Potts & Van Wassenhove [1991] compare descent and simulated annealing algorithms with other special-purpose heuristics for $1 \| \sum T_j$ and $1 \| \sum w_j T_j$. For descent and simulated annealing, they use the swap neighborhood and impose a strict limit on run times. The descent method performs surprisingly well relative to simulated annealing for both the unweighted and weighted problems if moves to solutions with the same objective function value are accepted (which commonly occur if two early jobs are interchanged). For $1 \| \sum T_j$, however, a special-purpose decomposition heuristic is slightly superior. On the other hand, a finely tuned simulated annealing algorithm that employs a descent routine gives the best results for $1 \| \sum w_j T_j$.

Crauwels, Potts & Van Wassenhove [1996] propose various local search algorithms for $1 \| \sum w_j T_j$. For the natural sequence representation, the swap neighborhood is used in multistart descent, simulated annealing, threshold accepting and taken search methods. A novel partition representation uses binary strings, where each element indicates whether the corresponding job is on time or late. To convert the partition representation into a sequence, a procedure is used which first sequences those jobs that are indicated as on time in EDD order. Any job that cannot be completed by its due date in this EDD sequence is removed, and is subsequently regarded as late. Then, where possible, late jobs are scheduled between the on-time jobs, provided this does not make them finish before their due dates, and finally any remaining late jobs are scheduled at the end of the sequence in SWPT order. Lastly, descent is applied to the resulting sequence, where the descent method uses the transpose neighborhood of Section 4.1. For the partition representation, neighborhood search algorithms use the reassign neighborhood and a genetic algorithm uses a modified version of the recombination operator of Section 4.2. Computational tests for problems with up to 100 jobs are used to compare the performance of the algorithms. The best-quality solutions are obtained with the genetic algorithm that uses the partition representation, and with the two tabu search algorithms that use the sequence and partition representations.

5.4 Weighted number of late jobs

To specify a solution of the problem $1 \| \sum w_j U_j$, it is sufficient to partition the jobs into two subsets: those which are on time and those which are late. A schedule is constructed from the partition by sequencing first the on-time jobs in EDD order, then the late jobs are sequenced arbitrarily after all on-time jobs. For $1 \| \sum U_j$ an algorithm of Moore [1968] solves the problem in $O(n \log n)$ time: jobs are added in EDD order to the end of a partial schedule of on-time jobs, and if the addition of job j results in it being completed after time d_j, a job in the partial schedule with the largest processing time is removed and deemed to be late.

The problem $1 \| \sum w_j U_j$ is shown by Karp [1972] to be NP-hard in the ordinary sense. However, it is pseudopolynomially solvable, since Lawler & Moore [1969] propose a dynamic programming algorithm that requires $O(n \sum_j p_j)$ time. Potts & Van Wassenhove [1988] present an $O(n \log n)$ procedure that solves the linear programming relaxation of a natural integer programming formulation of $1 \| \sum w_j U_j$. The resulting lower bounding scheme has two uses: first in reduction tests that are designed to eliminate jobs from the problem, and second in a branch-and-bound algorithm. In comparative computational tests, Potts & Van Wassenhove solve problems with up to 1000 jobs using the algorithm of Lawler & Moore and with their branch-and-bound algorithm.

Since $1 \| \sum w_j U_j$ is a partitioning problem, the local search approaches of Section 4.2 are applicable. However, an arbitrary partition may not yield a feasible subset of on-time jobs, so it is appropriate to suggest a repair/translation procedure. We propose the following method which is a variant of Moore's algorithm. If some job j is completed after time d_j, selected jobs in the partial schedule up to and including job j are removed and deemed to be late until either job j is on time or j itself is removed: at each stage the job k to be removed is chosen so that p_k/w_k is as large as possible.

5.5 Total weighted earliness and tardiness

Most research that considers earliness as an optimality criterion also includes a tardiness component. A thorough survey of this area of scheduling is given by Baker & Scudder [1990]. One widely studied model is $1 | d_j = d | \sum (w_j E_j + w'_j T_j)$, in which jobs have a common due date d. For the case of unit weights, Hall, Kubiak & Sethi [1991] show that this problem is NP-hard in the ordinary sense, and they propose an $O(n \sum_j p_j)$ pseudopolynomial dynamic programming algorithm. As indicated in Section 4.2, Baker & Scudder show that the following results of Kanet [1981] generalize to the case of arbitrary weights: there is no idle time between jobs, and jobs that are completed no later than time d are sequenced in LWPT order, whereas jobs completed after time d are sequenced in SWPT order. This latter result is known as the V-shaped property. If $d \geqslant \sum_j p_j$, Kanet derives an $O(n \log n)$ algorithm for $1 | d_j = d | \sum (E_j + T_j)$ by showing that the job in position $\lceil n/2 \rceil$ completes at time d, and by assigning jobs in nonincreasing order of processing times alternately to the first and last unfilled position in the

sequence. Closely related to these earliness–tardiness problems is $1 \| \sum (C_j - \bar{C})^2$, where $\bar{C}$ is the average completion time, for which the objective is to minimize the completion time variance. Eilon & Chowdhury [1977] show that the V-shaped property holds in this case, although this result does not extend to $1 \| \sum w_j (C_j - \bar{C})^2$ [Cai, 1995]. Moreover, these problems are NP-hard in the ordinary sense, but $1 \| \sum (C_j - \bar{C})^2$ is pseudopolynomially solvable [Kubiak, 1993], and $1 \| \sum w_j (C_j - \bar{C})^2$ is open with respect to pseudopolynomial solvability.

Any problem in which the V-shaped property holds can be treated as one of partitioning. Thus, the approaches of Section 4.2 are applicable. Mittenthal, Raghavachari & Rana [1993] propose a simulated annealing algorithm which is applicable when the V-shaped property holds. Adopting the natural representation, they use the reassign neighborhood and a restricted swap neighborhood in which the only jobs that are eligible for interchange are adjacent pairs in an SPT ordering. Their algorithm first applies two descent procedures, the first of which uses the reassign neighborhood and the second uses the restricted swap neighborhood, and then applies simulated annealing with the restricted swap neighborhood. Computational results for $1 \| \sum (C_j - \bar{C})^2$ with instances containing up to 20 jobs show that the algorithm generates an optimal solution for each test problem. Lee & Kim [1995] propose a parallel genetic algorithm for $1 | d_j = d | \sum (w_j E_j + w'_j T_j)$ which also uses the natural representation for partitioning problems. Note, however, that the job identified to finish before or at time d may actually be completed after time d when a schedule is constructed, and vice versa. Rather than a standard mutation operator, Lee & Kim suggest recalculating the binary values in the string according to which jobs are actually late. Parallel implementation is carried out by arranging a number of subpopulations in a ring. Each subpopulation evolves independently of the others, except that the best solution in each subpopulation is communicated to each of the two neighboring populations. Computational results with the genetic algorithm, using population sizes that vary between 100 and 200, show that the average deviation of the solution value from the optimum is less than 0.25% for 40-job problems. Gupta, Gupta & Kumar [1993] propose a genetic algorithm for $1 \| \sum w_j (C_j - \bar{C})^2$ in which they adopt a natural representation of solutions as a sequence and use the partially matched crossover described in Section 4.1.

For $1 \| \sum (w_j E_j + w'_j T_j)$ an optimal schedule may include machine idle time between jobs. However, for a given processing order of jobs, procedures of Garey, Tarjan & Wilfong [1988] and Davis & Kanet [1993] can be used to construct an optimal schedule. Thus, solutions can be represented as sequences, and the local search methods of Section 4.1 are applicable. Yano & Kim [1991] adopt this approach by using the transpose neighborhood in their descent algorithm. Computational results for problems with up to 20 jobs in which the weights are proportional to the processing times of the respective jobs indicate that their algorithm consistently generates an optimal solution.

Woodruff & Spearman [1992] propose a tabu search approach for a variant of the problem $1 | \bar{d}_j, s_{fg} | \sum w_j E_j + \sum c_{fg}$, in which there are general family setup

times and costs. In this variant there is a selection process whereby some jobs are eligible for rejection, although this incurs a cost. By not allowing machine idle time, a representation of solutions as a sequence is adopted. The insert neighborhood is used, and a translation mechanism is applied to reject those eligible jobs that are completed after their deadlines.

6 PARALLEL-MACHINE PROBLEMS

In this section, we consider the application of local search to the scheduling of identical, uniform and unrelated parallel machines. As in Section 5, we review the main results and, where possible, give an indication of problem 'hardness' based on the performance of enumerative algorithms. Our discussion excludes the extensive research on worst-case analysis of approximation algorithms.

In our description of local search methods, it is convenient to picture a two-stage solution procedure: jobs are assigned to machines in the first stage, and each individual machine is scheduled in the second stage. For some problems, such as $P \| C_{max}$, $Q \| C_{max}$, and $R \| C_{max}$, this second stage is trivial since job sequences do not affect the objective function. Other problems permit a simple polynomial algorithm for the second stage, so they can be viewed as pure assignment problems. Examples for unrelated parallel machines (which also hold for the corresponding problems with identical and uniform machines) include $R \| L_{max}$, for which the EDD rule solves the single-machine problems, $R \| \sum w_j C_j$, for which the SWPT rule is used, and $R \| \sum U_j$, for which Moore's algorithm is used. For a third category, which includes $R \| \sum w_j T_j$, combined assignment and sequencing decisions are necessary.

For the pure assignment problems, the approaches given in Section 4.2 can be applied; Section 4.3 provides guidelines for combined sequencing and assignment problems. Our discussion is restricted to natural representations of solutions; we are not aware of any research on parallel-machine scheduling problems in which other representations are used.

6.1 Identical parallel machines

A generalization by Conway, Maxwell & Miller [1967] of the SPT rule allows the problem $P \| \sum C_j$ to be solved in $O(n \log n)$ time. For other optimality criteria, the situation is gloomy. Bruno, Coffman & Sethi [1974] and Lenstra, Rinnooy Kan & Brucker [1977] show that $P2 \| \sum w_j C_j$ and $P2 \| C_{max}$ are NP-hard in the ordinary sense; this implies the NP-hardness of $P2 \| L_{max}$ and $P2 \| \sum U_j$.

Finn & Horowitz [1979] propose a descent method for $P \| C_{max}$ which requires $O(n \log m)$ time. Jobs are reassigned from a most heavily loaded to a least heavily loaded machine. A more sophisticated descent algorithm is suggested by França et al. [1994]. They use a combined critical reassign and critical swap neighborhood, as described in Section 4.2. A partitioning of jobs according to the similarity of their processing times is useful in selecting neighborhood moves. Hübscher & Glover [1994] propose a tabu search algorithm for $P \| C_{max}$.

They replace the original maximum completion time objective function by $\sum_{i=1}^{m} (\bar{H}_i - \bar{H})^2$, where H_i is the total processing time assigned to machine i and $\bar{H} = \sum_{j=1}^{n} p_j/m$ represents the ideal machine load. A combined critical reassignment and critical swap neighborhood is used, where the job to be reassigned, or one of the jobs to be swapped, is moved from the most heavily loaded machine to a machine with load less than $\bar{H}$. After a job is moved from a machine, a tabu list entry prevents a job with the same processing time from moving to this machine. However, some tabu list entries are periodically activated and deactivated with the aim of diversifying and intensifying the search. A further special feature is the use of 'influential diversification', which is a device that modifies the current solution when no improvement in the best solution is observed for a long period. Here two machines are selected that have a surfeit and deficit of long jobs, and these long are redistributed uniformly between this pair of machines. Computational results for problems with up to 50 machines and 2000 jobs indicate that the algorithm generates solutions with a makespan that is very close to $\bar{H}$, and that influential diversification is beneficial in detecting these solutions.

Among the various branch-and-bound algorithms for $P \| \sum w_j C_j$, the approach adopted by Belouadah & Potts [1994] in which lower bounds are obtained by a Lagrangean relaxation of machine capacity constraints appears best; problems with up to 20 jobs and 8 machines can be solved. Barnes & Laguna [1993] use the combined reassign and swap neighborhood of Section 4.2 in a tabu search algorithm for $P \| \sum w_j C_j$. By storing partial sums of processing times and weights on each machine in the current schedule, the total weighted completion time of a neighbor is evaluated in constant time. A tabu list of length 8 contains jobs which are forbidden to move from their current machine assignments; the job changing its machine assignment in a reassign move, or one of the jobs changing its assignment in a swap move, is stored at each iteration. In an attempt to diversify the search, swap moves are forbidden (except for the first few iterations) if they do not reduce the total weighted completion time. Computational results indicate that the algorithm consistently generates optimal solutions for problems with up to 4 machines and 30 jobs.

The problem $P|\bar{d}_j| \sum w_j E_j$ is equivalent to $P|r_j| \sum w_j C_j$ provided that the deadlines are sufficiently large. Laguna & González Velarde [1991] propose a tabu search algorithm for the former problem, although our description of their work refers to $P|r_j| \sum w_j C_j$. For this combined assignment and sequencing problem, they use a combination of the transpose, swap, and insert neighborhoods described in Section 4.3. To limit the size of the neighborhood, two jobs may be swapped or one job inserted after another job, only if the pair of jobs have similar release dates. Their tabu list, which they suggest should have length $\lfloor \sqrt{n} \rfloor + 5$, contains jobs that are forbidden to move under a transpose, swap or insertion. After an insert move is executed, the corresponding job is placed on the tabu list, whereas after a transpose or swap move, the job experiencing the larger reduction in weighted completion time is stored. A special feature when no improving move is possible is that job j is forced to move, where j is chosen to have

the largest weighted completion time among jobs that are not on the tabu list. The authors suggest that, when the tabu search algorithm terminates, a branch-and-bound algorithm for $1|r_j|\sum w_jC_j$ can be applied to each of the m machines to optimally schedule the jobs currently assigned to those machines.

Hou, Ren & Ansari [1990] report some preliminary work on a genetic algorithm for $P|prec|C_{max}$. They use a set of strings, one for each machine, to represent a solution. They also restrict solution strings by using a 'height' concept based on the maximum length chain of predecessors for a job. This is also used in the crossover mechanism to ensure that only feasible solutions are generated. A similar approach is used by Hou & Li [1991] for a related problem arising in the scheduling of flexible manufacturing systems in which the transport system between machines (using automated guided vehicles) must also be scheduled.

Motivated by the problem $P\|C_{max}$, Hellstrom & Kanal [1992] propose a neural network approach which attempts to find a schedule in which all jobs are completed by a given threshold value. The output of a neuron determines whether or not a job is assigned to a particular machine.

6.2 Uniform parallel machines

A further refinement by Horowitz & Sahni [1976] to the generalized SPT algorithm for $P\|\sum C_j$ allows the problem $Q\|\sum C_j$ to be solved in $O(n \log n)$ time. The NP-hardness for other optimality criteria follows from the corresponding results for identical parallel machines.

Lo & Bavarian [1992] compare simulated annealing algorithms for $Q|\bar{d}_j|C_{max}$. The deadline constraints are enforced using a penalty function approach. The problem is treated as one of combined assignment and sequencing, and the insert neighborhood is used.

6.3 Unrelated parallel machines

The problem $R\|\sum C_j$ is formulated by Horn [1973] and Bruno, Coffman & Sethi [1974] as a weighted bipartite matching problem; hence it is solvable in $O(n^3)$ time. As for uniform machines, the NP-hardness results given in Section 6.1 for identical machines imply NP-hardness for the corresponding unrelated machine problems.

Using a lower bound based on a surrogate relaxation, the branch-and-bound algorithm of Van de Velde [1993] solves instances of $R\|C_{max}$ with up to 200 jobs and four machines. Hariri & Potts [1991] propose a descent algorithm for $R\|C_{max}$ in which both the critical reassign and critical swap neighborhoods of Section 4.2 are used. With a suitable implementation, the objective function value of a neighbor can be computed in constant time. Computational results of Hariri & Potts indicate that this descent algorithm generates better-quality solutions than various two-phase heuristics [Potts, 1985b; Lenstra, Shmoys & Tardos, 1990], which use linear programming in their first phase to schedule most of the jobs. Hariri & Potts also report on initial experiments which indicate

that a more complicated neighborhood structure has little effect on solution quality. Glass, Potts & Shade [1994] propose simulated annealing and tabu search algorithms based on the critical reassign and critical swap neighborhoods. They also describe a genetic algorithm which uses the natural recombination operators suggested in Section 4.2, and a corresponding genetic descent algorithm in which the descent algorithm of Hariri & Potts is applied to each solution in every population. Computational results show that the standard genetic algorithm performs poorly relative to the other three algorithms which are roughly comparable. These results are presented in Section 8.2.

Kanet & Sridharan [1991] consider the problem $R|r_j, t_{jk}|\sum f_j(C_j)$ in which the objective function is an arbitrary convex function of job completion times. They report preliminary work using a start time representation in a genetic algorithm with the recombination operator which is described in Section 4.3.

7 MULTI-STAGE PROBLEMS

In this section we consider the application of local search to flow shop, open shop, and job shop problems. As is the case in most studies, the main focus of our discussion is on problems with the maximum completion time objective. Many local search algorithms for $F\|C_{\max}$, $O\|C_{\max}$ and $J\|C_{\max}$ are easily adapted to other optimality criteria. We follow the same format as in the two previous sections by reviewing the complexity results and commenting on problem 'hardness' by referring to the performance of the best currently available branch-and-bound algorithms.

Permutation flow shops in which each machine processes the jobs in the same order are sequencing problems, so Section 4.1 provides guidelines for the design of local search methods. More generally, flow shops, open shops, and job shops are multisequencing problems to which the approaches of Section 4.4 can be applied.

7.1 Flow shops

For $F\|C_{\max}$ Conway, Maxwell & Miller [1967] observe there exists an optimal schedule with the same processing order of jobs on the first pair of machines and the same order on the last pair of machines. Thus, for $F2\|C_{\max}$ and $F3\|C_{\max}$, it is sufficient to consider permutation schedules in which each machine processes the jobs in the same order. Johnson [1954] gives an $O(n \log n)$ algorithm for $F2\|C_{\max}$: the jobs with $p_{1j} \leqslant p_{2j}$ are sequenced first in nondecreasing order of p_{1j}; the remaining jobs are then sequenced in nonincreasing order of p_{2j}. Garey, Johnson & Sethi [1976] show that $F3\|C_{\max}$ is strongly NP-hard. Although permutation schedules are not guaranteed to provide optimal solutions for $Fm\|C_{\max}$ when $m \geqslant 4$, we follow a tradition in the literature and henceforth concentrate on the permutation flow shop, i.e., finding the best permutation schedule.

The best available branch-and-bound algorithms are those of Lageweg, Lenstra & Rinnooy Kan [1978] and Potts [1980], which compute lower bounds by solving two-machine subproblems using Johnson's algorithm. Their performance is not entirely satisfactory, however: they experience difficulty in solving instances with 15 jobs and 4 machines. Strong NP-hardness for $F2 \| L_{\max}$ and $F2 \| \sum C_j$ is established by Lenstra, Rinnooy Kan & Brucker [1977] and Garey, Johnson & Sethi, respectively, and NP-hardness in the ordinary sense for $F2|s_f|C_{\max}$ is established by Kleinau [1993] for the case of an arbitrary number of families F.

For heuristics that do not employ neighborhood search, the $O(mn^2)$ insertion method of Nawaz, Enscore & Ham [1983] is best for the permutation flow shop $F \| C_{\max}$; Taillard [1990] gives a derivation of its complexity. The insertion method builds a sequence by adding a job in the best position to the current partial sequence until it produces a complete sequence. Rather surprisingly, it outperforms the descent algorithm of Dannenbring [1977], which is based on the transpose neighborhood.

Various local search methods are available for the permutation flow shop $F \| C_{\max}$. They each use a natural representation of solutions as a sequence. Simulated annealing algorithms are proposed by Osman & Potts [1989] and by Ogbu & Smith [1990]. Both approaches compare the insert and swap neighborhoods; the insert neighborhood performs better in computational tests. Ogbu & Smith attribute this to the comparatively large size of the neighborhood, whereas Osman & Potts suggest it may depend upon the objective function. In the algorithm of Ogbu & Smith, the probability of accepting a worse solution is independent of the objective function values, whereas Osman & Potts use a standard acceptance probability. An evaluation of these two approaches by Ogbu & Smith [1991] finds they give similar results, although the algorithm of Osman & Potts is marginally more effective.

Tabu search algorithms are proposed by Widmer & Hertz [1989], Taillard [1990], Reeves [1993b], and Nowicki & Smutnicki [1996b]. Widmer & Hertz use the swap neighborhood, adopt a tabu list of length 7 that prevents either of the interchanged jobs returning to its previous position, and search the complete neighborhood before making a move to the best neighbor (a best improve acceptance strategy). Taillard suggests an improvement to each of the key components in the method of Widmer & Hertz. Based on computational tests, he claims that the insert neighborhood is better than swap, that a tabu list containing values of $C_{\max}$ is superior to one storing forbidden positions for jobs, and that a first improve acceptance strategy making the first move which reduces the maximum completion time is slightly better than the best improve acceptance strategy. Taillard also derives a method for evaluating all insert neighbors in $O(mn^2)$ time (although to evaluate any one of the $(n-1)^2$ neighbors requires $O(mn)$ time). Reeves proposes the use of a restricted version of the insert neighborhood in which a specific subset of jobs is eligible for insertion at each iteration. The best computational results are obtained for subsets containing six jobs that are created by a random partition of the original set of jobs. After all

subsets are considered, new subsets are created by random partitioning. The algorithm has a tabu list of length 7 that prevents the inserted job returning to its previous position, and uses a best improve acceptance strategy on the restricted neighborhood.

Nowicki & Smutnicki's [1996b] tabu search algorithm uses a restricted version of the insert neighborhood, where the block structure of some critical path (as defined in Section 4.4) determines the eligible neighbors. A job without an operation that begins or ends a block may be inserted in the position corresponding to the end of its block and in the following few positions, or in the position corresponding to the start of its block and in the previous few positions. Alternatively, the eligible positions for insertion of a job which has an operation that ends one block and an operation that starts another block are as follows: the end of the block that it starts and the following few positions, and the start of the block that it ends and the previous few positions. A tabu list of length 8 contains precedence constraints, which are created as follows. For two adjacent jobs in the original sequence, a tabu list entry defines the second job to be a predecessor of the first if either the second of these jobs is inserted in an earlier position or the first job is inserted in a later position. Using ideas from the analysis of Taillard [1990], together with the observation that the block structure limits the number of positions to which a job is allowed to more, Nowichi & Smutnicki establish that all neighbors can be evaluated in $O(m^2n)$ time. Another special feature of their algorithm is a backtracking procedure: when an iteration limit is reached, the procedure is restarted from the best solution, but with a different neighborhood move to that made previously.

Computational results of Reeves [1993b] indicate that his tabu search algorithm is superior to the simulated annealing method of Osman & Potts and also to the corresponding tabu search algorithm which uses the full insert neighborhood. The computational results of Nowicki & Smutnicki [1996b] show that their algorithm generates better-quality solutions than those of Taillard or Werner [1993] (see below for a description of Werner's path algorithms) and it requires much less computation time.

Werner [1993] proposes a class of 'path' algorithms for the permutation flow shop $F \| C_{\max}$. Each algorithm can be viewed as descent with exploration and backtracking. A restricted version of the insert neighborhood is used: a neighbor is not considered if it allows the possibility that all critical operations in the current solution are to remain critical and consequently cannot improve upon the objective function value. A sequence of restricted insert moves is performed (they are accepted irrespective of objective function values). If a solution is found which is better than all of those that are previously generated, a new sequence of restricted insert moves is initiated from this solution. On the other hand, after generating a prespecified number of moves, if no improved solution is found, the algorithm backtracks to the best solution and another sequence of restricted insert moves is made. The process for selecting a neighbor is quite complicated, and is based on quickly computed lower bounds for the maximum completion time. Stochastic versions of the algorithm perform a move to the

best of a random selection of restricted insert neighbors, and backtracking may occur when a probabilistic condition is satisfied, rather than after a prespecified number of moves. Upon termination of this procedure, a descent algorithm based on the block transpose neighborhood is applied. Computational results indicate that a stochastic version of this algorithm yields the best-quality solutions, and this algorithm is superior to the simulated annealing and tabu search methods of Osman & Potts and Widmer & Hertz.

Reeves [1995] proposes a genetic algorithm which uses the reorder crossover. In comparative computational results which have the same computation time limit for each algorithm, the genetic algorithm produces better-quality solutions than multistart descent and is comparable with the simulated annealing method of Osman & Potts.

Several of the algorithms for permutation flow shop $F \| C_{\max}$ are tested on the problems generated by Taillard [1993b], thus enabling some additional comparisons to be made. From the various sets of computational results, we conclude that the tabu search algorithms of Reeves and Nowicki & Smutnicki generate the best quality solutions: they appear to outperform the simulated annealing algorithm of Osman & Potts, Reeves' genetic algorithm, and Werner's path algorithms. The current champion is the method of Nowicki & Smutnicki, which exhibits superiority over Reeves' tabu search algorithm.

We now discuss research which assumes different optimality criteria. Kohler & Steiglitz [1975] compare different neighborhoods in descent algorithms for $F2 \| \sum C_j$, and Krone & Steiglitz [1974] propose a descent algorithm for $F \| \sum C_j$ in which permutation schedules are not assumed. Based on the insert neighborhood, the algorithm of Krone & Steiglitz searches for the best permutation schedule, and then attempts insertions on the individual sequences defining the processing orders on the different machines. For the permutation flow shop $F \| \sum w_j C_j$, Glass & Potts [1996] perform a computational comparison of multistart descent, simulated annealing, threshold accepting, tabu search, and two genetic algorithms, one of which applies descent to each solution in every population. The neighborhood search algorithms each use the swap neighborhood, which performs marginally better than insert in initial experiments. Also, based on further initial tests, the insertion crossover is used in the first genetic algorithm, whereas the genetic descent algorithm employs the reorder crossover. Simulated annealing and the genetic algorithm that incorporates descent generate the best-quality solutions, and the genetic descent algorithm is slightly superior. These results are presented in Section 8.3.

In independent studies by Kim [1993] and Adenso-Días [1992], the tabu search algorithm of Widmer & Hertz is adapted to the permutation flow shop problems $F \| \sum T_j$ and $F \| \sum w_j T_j$. In the algorithm of Adenso-Días, the swap neighborhood is used in the first part of the search; thereafter, insert neighbors only are considered. To save on computation time, the complete neighborhood is not searched: a swap is allowed only if the positions of the two jobs are closer than a threshold value, and a job may only be reinserted in a position that is closer than a threshold value to its previous location in the sequence. The

threshold value decreases as the search progresses from an initial value of $n-1$ to a final value of 4. Computational results indicate that restricting the neighborhood reduces computation time, without affecting average solution quality. Cleveland & Smith [1989] compare genetic algorithms for a variant of the problem $F \| \sum (E_j + T_j^2)$ in which there are identical parallel machines at each stage. The problem is regarded as one of sequencing jobs at the first stage: jobs are processed in order of arrival at subsequent stages.

The permutation flow shop problem $F|s_f|C_{\max}$ is the subject of two studies. Vakharia & Chang [1990] represent solutions as sequences and propose a simulated annealing algorithm that uses a neighborhood based on the transpose of adjacent batches and of adjacent jobs. A genetic algorithm of Whitley, Starkweather & Shaner [1990] uses a recombination operator, called edge recombination, which is designed for use with the traveling salesman problem. They perform computational tests on problems with six machines in which all jobs from the same family are identical.

Various conclusions can be drawn from some of the studies on neighborhood search for the permutation flow shop problem. First, insert is the best neighborhood when minimizing the maximum completion time. Second, it is often advantageous to restrict the search to a subset of the neighbors. Moreover, the critical operations provide a useful guide as to which neighbors should be avoided. Third, an efficient computation of the exact or estimated objective function value for a neighbor has a substantial effect on the efficiency of the neighborhood search algorithm.

7.2 Open shops

Gonzalez & Sahni [1976] derive an elegant $O(n)$ algorithm for $O2 \| C_{\max}$, and show that $O3 \| C_{\max}$ is NP-hard in the ordinary sense. We are not aware of any literature on enumerative algorithms for $O \| C_{\max}$. Strong NP-hardness of $O2 \| L_{\max}$ and $O2 \| \sum C_j$ is established by Lawler, Lenstra & Rinnooy Kan [1981, 1982] and Achugbue & Chin [1982], respectively.

A problem that is related to the open shop occurs when scheduling jobs for which a number of process plans are available. The process plan for a job specifies which machines may carry out each operation and the precedence relations that exist between operations. Detailed process plans may also include choices on the tools to be used for particular operations (which will affect setup times). An open shop thus corresponds to a problem in which there are no precedence relationships between operations and in which each operation can only be performed by one machine. Bagchi et al. [1991] describe a genetic algorithm for the scheduling problem with process plans in which the partially matched crossover is used to handle the priority order between jobs and a type of uniform crossover is used for the machine and process plan selection. Husbands, Mill & Warrington [1991] discuss a genetic algorithm in which there are separate populations of process plans for each part to be made, and also a population of priority orders that define

a schedule. Each population evolves at the same time, and the individual process plans are evaluated by simulating their effect when run in conjunction with other process plans and a particular priority order.

7.3 Job shops

The problem $J2 \| C_{max}$, and consequently $J2 \| L_{max}$, is shown by Garey, Johnson & Sethi [1976] to be strongly NP-hard. Strong NP-hardness of $J2 \| \sum C_j$ is implied by the corresponding result for the two-machine flow shop. Various enumerative and heuristic methods for $J \| C_{max}$ employ the following *disjunctive graph* formulation. For each operation O_{ij}, there is a vertex with weight p_{ij}. The precedence between each pair of consecutive operations on the same job is represented by a directed arc. And there is an undirected edge corresponding to each pair of operations that require the same machine. Choosing a processing order on every machine corresponds to orienting the edges to produce a directed acyclic graph. It is therefore required to find an orientation that minimizes the length of a longest or critical path, where the length is defined as the sum of weights of vertices that lie on the path.

The job shop problem $J \| C_{max}$ is regarded as one of the hardest in combinatorial optimization. For example, a classic 10-job 10-machine instance that is originally given by Fisher & Thompson [1963] has only fairly recently been solved to optimality by Carlier & Pinson [1989], Applegate & Cook [1991], and Brucker, Jurisch & Sievers [1994]. Within their algorithms these authors each consider single-machine subproblems which are obtained by relaxing all edges in the disjunctive graph that are not yet oriented, except those corresponding to operations on some selected machine. The resulting problem is equivalent to $1|r_j|L_{max}$, where all due dates are nonpositive. For each operation on the selected machine in this single-machine subproblem, its release date is the length of a longest path to any predecessor of this operation, and its due date is minus the length of a longest path from any sucessor.

Many heuristics are based on the use of priority rules, which are surveyed by Haupt [1989]. Such approaches use a priority rule to select an operation from a set of candidates to be sequenced next. The candidates may be chosen to create a nondelay schedule in which no machine idle time is allowed if operations are available to be processed. There is no guarantee of an optimal solution that is a nondelay schedule, so it may be preferable to generate an active schedule: Giffler & Thompson [1960] propose a procedure for active schedule generation. A limited delay schedule offers a useful compromise between nondelay and active schedules. Although priority rule heuristics are undemanding in their computational requirements, the quality of schedules that are generated tends to be erratic.

An effective heuristic method is the *shifting bottleneck* procedure of Adams, Balas & Zawack [1988]. It is based on the observation that a schedule can be constructed by selecting each machine in turn and orienting all of the corresponding edges in the disjunctive graph formulation. The problem of orienting these

edges is equivalent to solving $1|r_j|L_{max}$, where previously oriented edges are used in the computation of release dates and due dates: the branch-and-bound algorithm of Carlier [1982] computes a solution. The unscheduled (bottleneck) machine is selected so that the maximum lateness for the corresponding problem $1|r_j|L_{max}$ is as large as possible. After the scheduling of a bottleneck machine by orienting the corresponding edges, the machine reschedule neighborhood of Section 4.4 is searched in an attempt to find an improved partial schedule (where each neighbor requires the solution of $1|r_j|L_{max}$ by Carlier's algorithm). When all edges are oriented, a final step of the procedure searches for a further improvement by considering one k-machine reschedule neighbor, where $k \leqslant \sqrt{m}$, for which a sequence of k instances of $1|r_j|L_{max}$ are solved.

Adams, Balas & Zawack also propose an enumerative version of their procedure in which several alternatives are considered for the machine that is to be scheduled next. Various improvements to the original shifting bottleneck procedure have been suggested. Applegate & Cook [1991] propose variants (Bottle-k for $k = 4, 5, 6$) in which, for the last k machines, every possibility is considered when selecting the machine to be scheduled next. In another procedure (Shuffle), they use their branch-and-bound procedure to generate a k-machine reschedule neighbor of the schedule obtained using Bottle-5, where k is quite large (for example, $k = 15$ for $m = 20$).

Dauzère-Pérès & Lasserre [1993] observe that the current orientation of some edges in the shifting bottleneck procedure may create a path between two operations that require the same machine. In this case the minimum time delay between the start times of these operations can be computed. Thus, the $1|r_j|L_{max}$ problems that are considered within the shifting bottleneck procedure should ideally incorporate *delayed precedence constraints* to account for these delays. Dauzère-Pérès & Lasserre use a heuristic approach for these single-machine problems with delayed precedence constraints, whereas Balas, Lenstra & Vazacopoulos [1995] obtain an exact solution by designing a generalized version of Carlier's algorithm. The quality of the solution obtained from the basic shifting bottleneck procedure of Adams, Balas & Zawack is relatively poor, although its computational requirements are modest. Its variants each exhibit an improved performance at the expense of a greater investment in computation time.

Various local search algorithms for $J\|C_{max}$ have been proposed, and they are reviewed by Vaessens, Aarts & Lenstra [1996]. The neighborhood search approaches use a natural representation, but with an exclusion mechanism of the type outlined in Section 4.4.

Simulated annealing algorithms are the subject of three studies for $J\|C_{max}$. First, Van Laarhoven, Aarts & Lenstra [1992] suggest the use of the critical transpose neighborhood. On the other hand, Matsuo, Suh & Sullivan [1988] adopt the smaller critical end transpose neighborhood. However, they observe that complex precedence relations between operations propagate the effect of a local change throughout the entire job shop system, and hence a neighborhood move may lead to deterioration of the objective function value, even when a superior solution can be obtained by a small number of further moves. Thus, if

a neighbor yields an inferior solution, further transposes are attempted involving a predecessor or sucessor of one of the originally transposed operations. Yamada, Rosen & Nakano [1994] use the critical end insert neighborhood. Their algorithm backtracks to the best schedule that is currently generated whenever 3000 moves are accepted that do not improve the best solution value, and resets the temperature appropriately. Computational results in these three studies show that each of the simulated annealing algorithms generates better-quality solutions than those of the shifting bottleneck procedure and its variants. The method of Yamada, Rosen & Nakano finds better solutions than the other two algorithms, although at considerable computational expense. The best compromise between solution quality and computation time is offered by the algorithm of Matsuo, Suh & Sullivan.

Tabu search provides an attractive alternative to simulated annealing for $J \| C_{max}$. We describe four algorithms. Taillard [1994] adopts the critical transpose neighborhood that is used by Van Laarhoven, Aarts & Lenstra [1992]. Following the transpose of two operations, the tabu list forces these operations to be sequenced in adjacent positions. Special features of the algorithm include the replacement of exact evaluations of the maximum completion time of each neighbor by quickly computed lower bound estimates, a tabu list length that changes randomly after specified numbers of iterations are performed, and the use of a penalty function which aims to prevent the repeated transpose of an operation to earlier positions in the schedule.

Barnes & Chambers [1995] use the framework of Taillard's method to design an alternative tabu search algorithm. We highlight the special features of their approach. To enable the execution of a backtracking procedure, a schedule is stored if it has a better objective function value than any previously generated solution. When the algorithm fails to detect an improved solution after a specified number of iterations, it restarts using the best of the stored schedules as an initial solution. Several runs are performed from this initial schedule, each with a different tabu list length. Following these runs with a given selection of tabu list lengths, the corresponding initial schedule is removed from the collection of stored solutions, and further exploration is initiated from the best schedule that is currently stored.

Dell'Amico & Trubian [1993] use a composite neighborhood consisting of generalized critical end transpose – which allows the reordering of three critical operations (one of which must start or end a block) – and critical end insert. Their algorithm adopts the idea used by Taillard of selecting moves according to quickly computed lower bounds for the maximum completion time, rather than performing a full objective function evaluation of each neighbor. They also use a variable-length tabu list which prevents the reorientation of disjunctive arcs that are reversed in the previous moves.

Lastly, Nowicki & Smutnicki [1996a] use the critical end transpose neighborhood. Their tabu list also prevents the reversal of the most recent transpose moves. However, when there is no allowable move, the tabu list is updated by discarding the oldest entry and adding a duplicate of the most recent entry. As in

their tabu search algorithm for the permutation flow shop problem, Nowicki & Smutnicki incorporate a backtracking procedure: when an iteration limit is reached, the procedure is restarted from one of the best solutions, but with a different neighborhood move to that made previously. On the evidence of the computational results quoted by the different authors, the four tabu search algorithms are generally superior to simulated annealing and to variants of the shifting bottleneck procedure. The method of Nowicki & Smutnicki is clearly superior to the other three methods in terms of both solution quality and computation time.

Balas & Vazacopoulos [1994] propose a *guided local search* procedure, which resembles tabu search with backtracking. The critical end insert neighborhood is used to construct a sequence of 'neighborhood trees', where a feasible solution is associated with each node of a tree. For any node, each of its immediate descendants is a neighbor of the corresponding solution. Using lower bound estimates of the maximum completion time for each neighbor, a given number of nodes are created for the solutions with the lowest estimates. For a node that is created by inserting an operation before or after another operation, a precedence constraint is added to prevent the order of this pair of operations being reversed, and the immediate descendants that are created subsequently each have a precedence constraint which forces this pair to be sequenced in the same order as for the parent node. In addition to the restriction on the number of immediate descendants for each node, a limit is placed on the number of levels that are explored. After the creation of a neighborhood tree, one of the best solutions that is generated from a previous tree becomes the root node of a new tree.

Balas & Vazacopoulos also suggest several hybrids in which guided local search is embedded in the shifting bottleneck procedure. In the basic hybrid, instead of searching the machine reschedule neighborhood after a bottleneck machine is scheduled, a given number of neighborhood trees are generated using the current (partial) solution from the shifting bottleneck procedure at the root of the first tree. The best schedule from these neighborhood trees is then used for the continuation of the shifting bottleneck procedure. The other hybrids extend the basic version by allowing additional search. This further search is performed by removing the orientation of edges on certain machines, then constructing a new schedule using a combination of guided local search and shifting bottleneck routines. Computational results show that the guided local search procedure is superior to simulated annealing and variants of the shifting bottleneck procedure. The hybrid shifting bottleneck/guided local search methods produce high-quality solutions at reasonable compu- tational expense, and are preferred to pure guided local search. These hybrids also compare favorably with most of the tabu search algorithms, although it is not clear whether they are superior to the method of Nowicki & Smutnicki [1996a].

Genetic algorithms which use a variety of different representations have been proposed for $J \| C_{max}$. Several of these algorithms use a heuristic-based representation: a solution is constructed from its representation by applying some

heuristic method. Storer, Wu & Vaccari [1992] propose the data perturbation representation described in Section 3.1. A schedule is constructed from the perturbed data by using SPT as a priority rule to construct a limited delay schedule in which a small amount of unforced machine idle time is allowed. They also suggest a heuristic set representation: after partitioning the time horizon into time windows, each solution is represented by a string, where the entry in a particular position is the priority rule (chosen from a group of 6) used within the corresponding time window. Limited delay scheduling converts the representation into a schedule. Storer, Wu & Vaccari also consider a hybrid algorithm which uses a representation based on data perturbations and heuristic sets.

Dorndorf & Pesch [1995] implement a heuristic set representation in which a string entry is the priority rule (chosen from a group of 12) that is used in the corresponding iteration when creating an active schedule. A similar approach is used by Smith [1992]. Della Croce, Tadei & Volta [1995] propose the use of priority representation, an approach suggested previously by Davis [1985] and Falkenauer & Bouffouix [1991]. In this representation, each machine has a sequence that defines its priority order, and a type of insertion crossover is used in which the inserted subsection of a string is put into the same position that it previously occupied in the original string. Another limited delay scheduling procedure is used for generating a schedule from the representation. The algorithm performs better if the scheduling procedure is treated as a repair mechanism, so the priority representation for each machine is replaced by the corresponding processing order of operations in the schedule that is generated. A similar approach is used by Yamada & Nakano [1992] who use completion times to define priorities. For recombination, a set of candidate operations in an active schedule is found using the procedure of Giffler & Thompson [1960], and a random choice is made between the two candidates with the smallest completion time in each parent schedule. After a complete schedule is generated, this procedure is repeated with the same two parents. Dorndorf & Pesch design genetic algorithm based on the shifting bottleneck procedure. More precisely, using a representation that is a sequence of machines, a solution is constructed by using the representation to define the order in which the machines are scheduled within the shifting bottleneck procedure.

Three further genetic algorithms are proposed by Pesch [1993]. The first is based on the observations that a schedule can be constructed by selecting job pairs in turn, solving the corresponding two-job subproblem, and using the solution to orient the edges between operations on these jobs in the disjunctive graph formulation. The genetic algorithm is used to select the order in which the job pairs are considered. The second genetic algorithm of Pesch uses a representation that gives an (artificial) upper bound on each two-job subproblem. For each subproblem, the upper bound may allow some edges to be oriented on the basis of consistency tests. A complete schedule is constructed using a similar approach to that in the algorithm of Yamada & Nakano: the Giffler–Thompson procedure selects candidate operations, and a selection between candidates is

based on the two parent schedules. The third genetic algorithm of Pesch is similar to the second, but uses single-machine subproblems instead of two-job subproblems.

There are also genetic algorithms that do not rely on a heuristic-based representation. Nakano & Yamada [1991] use a type of ordered pair representation (see Section 4.1). They employ a standard crossover operation on the resulting binary strings (with the information corresponding to a single job pair for different machines making up a single string). A comparison of approaches based on translation and on repair (which they call 'forcing') shows repair to be more effective. Aarts et al. [1994] perform a computational comparison of several local search methods. Included in their comparison is a hybrid which incorporates descent into a genetic algorithm. Recombination in this genetic descent algorithm is achieved by choosing one of the two parents and then finding two operations that are eligible for transpose under the critical transpose neighborhood and that have the reverse order in the second parent. These jobs are then reversed in the first parent, and the whole process is repeated several times. A descent algorithm, which may use either the critical transpose or the extended version of the critical end transpose neighborhood, is applied to each newly generated solution. The performance of the genetic algorithms is variable. The most effective appears to be the third algorithm of Pesch [1993], which uses upper bounds on single-machine subproblems to orient edges. Although comparing favorably with simulated annealing, it cannot compete with the best tabu search method or the hybrid shifting bottleneck/guided local search algorithms.

Several of the algorithms for the job shop $J \| C_{\max}$ are tested on the problems generated by Fisher & Thompson [1963], Lawrence [1984], and Taillard [1993b]. Vaessens, Aarts & Lenstra [1996] collate the objective function values generated by the various algorithms and provide standardized computation times. These results are extended by Balas & Vazacopoulos [1994]. The candidates for champion algorithm are the tabu search algorithm of Nowicki & Smutnicki [1996a] and the hybrid shifting bottleneck/guided local search algorithms of Balas & Vazacopoulos. Section 8.4 presents the main findings of these computational studies that compare different approaches.

There are several neural network studies for the problem $J \| \sum C_j$. Foo & Takefuji [1988a, 1988b] propose a model in which the output of neurons defines precedences between operations. On the other hand, Foo & Takefuji [1988c] and Zhou et al. [1991] use a formulation in which the output of a neuron defines the starting time of the corresponding operation.

Some scheduling problems in flexible manufacturing systems have a job shop structure. Widmer [1991] proposes a tabu search algorithm for a variant of the problem $J \| C_{\max} + \sum w_j U_j$. Associated with each operation on every job is a requirement for certain tools to be loaded in a tool magazine that has limited capacity. The weighted number of reloadings of the tool magazine forms an additional component in the objective function. The algorithm uses a natural representation of solutions and employs the insert neighborhood.

In common with the permutation flow shop, there emerge some performance-enhancing features of neighborhood search algorithms for $J \| C_{\max}$. First, it is beneficial to restrict the neighborhood so that a critical operation is transposed, swapped, or inserted. Second, to use computation time efficiently, it may be preferable to estimate objective function values rather than perform an exact evaluation. Third, the possibility of backtracking to a previous solution, rather than always continuing the search from the current solution, allows the search to be redirected to parts of the solution space with a higher chance of finding a good-quality solution.

8 COMPUTATIONAL COMPARISONS

There are obvious similarities between different local search methods; even a genetic algorithm, which contains some unique features, will often be implemented with a pure neighborhood search component. Due to the scarcity of comparative computational studies for scheduling problems in the literature, it is not clear which of the various methods is the best. In this section we briefly summarize four computational studies in which different local search methods are compared against each other. These studies cover most of the different problem types that are introduced in Section 4.

When evaluating local search algorithms, there is clearly a trade-off between the investment in computation time and the quality of solution. To provide a fair comparison, the same computation time should be allocated to different methods; otherwise it is difficult to draw firm conclusions. It is usual to measure the effectiveness of a local search method by the relative (percentage) deviation of the value of the solution that is generated from the best known solution value (or from the best known lower bound if such a bound is sufficiently tight). These deviations are often averaged over several runs of the algorithm on the same problem or on different problems. Furthermore, maximum deviations are often used by themselves or in conjunction with averages to give an indication of the consistency of the heuristic in finding a near-optimal solution.

8.1 Merging ordered lists of jobs in $1|s_f|\sum w_jC_j$

Crauwels, Potts & Van Wassenhove [1997] describe a computational study for the problem $1|s_f|\sum w_jC_j$ in which F families of jobs with associated sequence-independent setup times are to be scheduled on a single machine to minimize the total weighted completion time. Recall that a solution procedure requires lists of jobs, each containing the jobs of a family in SWPT order, to be merged. The methods on which the computational comparison is performed are multistart descent (MD), simulated annealing (SA), threshold accepting (TA), tabu search (TS), and a genetic algorithm (GA), details of which are given below.

- *MD.* The multistart descent algorithm represents solutions as sequences and uses a type of block insert neighborhood, as proposed by Ahn & Hyun [1990],

which guarantees to maintain the jobs within each family in SWPT order. There are $\lfloor n/3 \rfloor$ restarts.

- *SA.* The simulated annealing method uses the same representation and neighborhood as in MD. The 'cooling' scheme follows a periodic pattern in which temperatures decrease and increase. Also, descent iterations are performed between each of these temperature changes. The method is applied to four different initial solutions, and the best solution is selected.
- *TA.* The threshold accepting method also uses the same representation and neighborhood as in MD. The method is implemented in a similar way to SA with periodic threshold values, and with descent iterations performed between changes of threshold values. The best solution obtained from four different starting solutions is chosen.
- *TS.* Tabu search is implemented with the same representation as in MD, but uses a subset of the shift neighborhood in which the shifted job can only be inserted between batches and the SWPT order within families is preserved. A tabu list of length 7 stores jobs that are forbidden to move. The best solution obtained from $\lfloor n/3 \rfloor$ different starting solutions is selected.
- *GA.* The genetic algorithm, proposed by Mason [1992], uses the batch-based representation of solutions described in Section 4.5. A further sophistication is introduced by noting that an optimal solution will contain no very small batches (in which the total processing is not sufficiently large relative to the setup). Whenever a new batch is started, a minimum batch size can be calculated and used to restrict the solutions that are considered. The population size is $2n$. The best solution from two independent runs is selected.

For different combinations of n and F, Crauwels, Potts & Van Wassenhove [1997] randomly generate 50 test problems. Computational results for problems in which setup times have the same distribution as processing times are summarized in Table 11.1. In particular, the number of problems (out of 50) for which an optimal solution is found (NO), the maximum relative deviation from an optimal solution value (MRD) expressed as a percentage, and the average computation time required in seconds (ACT) on an HP 9000/825 are listed for each algorithm.

It is seen from Table 11.1 that each of the algorithms MD, SA, TA, TS, and GA generates solutions of high quality. The best results, as measured by numbers of optimal solutions generated and maximum relative percentage deviations, are obtained using TS. A further advantage of TS is that it requires less computation time than the other methods; MD, SA, TA, and GA have similar performance.

8.2 Assignment of jobs in $R \| C_{\max}$

Glass, Potts & Shade [1994] give a computational comparison of local search algorithms for the problem $R \| C_{\max}$ of scheduling jobs on unrelated parallel machines to minimize the maximum completion time. A schedule is specified by

Table 11.1 Summary of results for $1|s_f|\sum w_jC_j$

		MD			SA			TA			TS			GA		
n	F	NO	MRD	ACT	NO	MRD	ACT	NO	MRD	ACT	NO	MRD	ACT	NO	MRD	ACT
30																
	4	49	0.35	0.7	50	0.00	1.0	47	0.35	1.1	50	0.00	0.3	46	0.47	1.3
	6	48	0.09	0.8	49	0.13	1.1	50	0.00	1.2	50	0.00	0.5	49	0.18	1.2
	8	49	0.25	0.9	50	0.00	1.2	48	0.25	1.3	49	0.25	0.8	50	0.00	1.2
	10	46	0.07	1.0	48	0.05	1.2	48	0.02	1.3	50	0.00	0.9	50	0.00	1.2
40																
	4	47	0.05	1.9	48	0.04	2.5	47	0.13	2.9	48	0.05	0.7	45	0.11	2.9
	6	49	0.02	2.2	49	0.32	2.8	49	0.03	3.1	48	0.05	1.1	48	0.17	3.0
	8	48	0.11	2.7	50	0.00	3.2	46	0.08	3.3	50	0.00	1.7	49	0.01	2.9
	10	46	0.09	2.5	48	0.14	2.8	45	0.07	3.0	49	0.00	2.1	49	0.03	2.6
50																
	4	46	0.26	3.9	48	0.04	4.8	47	0.03	5.5	50	0.00	1.2	40	1.42	5.7
	6	47	0.03	4.9	45	0.10	5.7	46	0.07	6.2	50	0.00	2.1	44	0.17	5.9
	8	43	0.08	5.5	46	0.21	6.2	45	0.03	6.3	47	0.05	3.1	46	0.08	5.5
	10	46	0.10	6.1	45	0.09	6.4	43	0.09	6.4	50	0.00	4.2	50	0.00	5.6
Average		47	0.13	2.8	48	0.09	3.2	47	0.10	3.5	49	0.03	1.6	47	0.22	3.3

NO = Number of problems (out of 50) for which an optimal solution is found.
MRD = Maximum relative deviation (%) of heuristic solution value from that of the optimum.
ACT = Average computation time in seconds on an HP 9000/825.

Table 11.2 Average relative deviations (%) for $R \| C_{max}$

		20 seconds					100 seconds		
m	n	DA	SA	TS	GA	GD	SA	TS	GD
5									
	20	3.1	0.1	0.8	5.1	0.0	0.0	0.8	0.0
	50	3.0	0.7	0.6	6.3	0.5	0.2	0.5	0.1
	100	1.3	0.8	0.5	4.4	0.7	0.6	0.4	0.2
10									
	20	6.6	0.1	0.9	15.0	0.4	0.0	0.9	0.1
	50	4.3	1.4	0.5	12.5	0.5	0.6	0.5	0.1
	100	2.7	1.5	0.2	10.3	1.1	0.5	0.1	0.7
20									
	20	5.5	0.1	0.4	12.7	0.4	0.0	0.4	0.2
	50	7.8	0.7	0.4	21.3	1.7	0.4	0.4	0.9
	100	3.4	2.7	0.3	20.7	2.2	0.8	0.2	1.3
50									
	20	5.8	0.0	0.5	16.2	0.0	0.0	0.5	0.0
	50	5.8	0.1	0.3	19.3	0.3	0.0	0.3	0.1
	100	5.7	0.4	0.4	30.2	1.8	0.2	0.3	1.2
Average		4.6	0.7	0.5	14.5	0.8	0.3	0.4	0.4

an assignment of jobs to machines. The study compares the following five algorithms:

- *DA.* The descent algorithm adopts a natural representation of solutions. The combined critical reassign and critical swap neighborhood described in Section 4.2 is used, and the algorithm terminates when a local minimum is detected.
- *SA.* The simulated annealing method uses the same representation and neighborhood as in DA. Another feature is the cooling schedule of Lundy & Mees [1986], which outperforms geometric cooling in initial tests.
- *TS.* Tabu search also uses the same representation and neighborhood as in DA. A tabu list of length 7 stores values of the maximum completion time.
- *GA.* The genetic algorithm uses the natural representation suggested in Section 4.2, in which each string is a list of machines to which the jobs are assigned. Recombination consists of performing a two-point crossover and mutating each element by changing the machine assignment of the corresponding job. The population size is 100.
- *GD.* The genetic descent algorithm is a variant of GA in which DA is applied to each new solution that is generated. The population size is 20.

The earliest completion time (ECT) heuristic of Ibarra & Kim [1977] provides the initial solution for the neighborhood search algorithms DA, SA, and TS.

Moreover, the initial populations in GA and GD are seeded with two copies of the ECT solution. For several solutions with the same maximum completion time, one with the smallest number of machines achieving the maximum machine load is generally preferred. This preference is incorporated into the acceptance rule for the neighborhood search algorithms.

Glass, Potts & Shade [1994] generate 20 test problems for various combinations of m and n, where some of the processing time matrices contain rows and columns that are correlated. The five algorithms are first run using a time limit of 20 seconds on an IBM 3090 computer. Then the more successful algorithms, SA, TS, and GD, are run using a time limit of 100 seconds. Average relative percentage deviations from the best known solution values are listed in Table 11.2.

Algorithm GA performs very poorly in this study, even compared with DA, which uses only a small proportion of the allowed computation time. However, when descent is incorporated into GA, we observe from the results for GD that the algorithm is competitive. This substantiates a widely held belief that, for a genetic algorithm to yield high-quality solutions, it needs to incorporate another heuristic or some problem-specific features. With the time limit of 20 seconds, TS performs marginally better than the other algorithms. However, both SA and GD improve substantially with additional run time, and the performances of SA, TS, and GD are comparable when the time limit is 100 seconds.

8.3 Sequencing of jobs in $F \| \sum w_j C_j$

Glass & Potts [1996] describe a computational study for the permutation flow shop problem $F \| \sum w_j C_j$, which requires the minimization of total weighted completion time. In contrast to the case of minimizing the maximum completion time, there is no efficient update mechanism for computing the objective function for a new solution generated by a local search algorithm. The study compares six local search algorithms, described below. In each case the natural representation of solutions as a sequence is adopted.

- *MD.* Multistart descent repeatedly applies descent, using different randomly generated initial sequences, until a prespecified time limit is exceeded. In initial experiments with the insert and swap neighborhoods of Section 4.1, swap yields slightly better results. Thus, the swap neighborhood is adopted.
- *SA.* The simulated annealing algorithm uses the swap neighborhood. Moreover, a geometric cooling schedule is used since it performs better than the scheme of Lundy & Mees [1986] in initial experiments.
- *TA.* Threshold accepting is implemented with the swap neighborhood, and the threshold value decreases linearly after each iteration. A quadratic decrement of threshold values produces inferior results in initial experiments.
- *TS.* The tabu search method also uses the swap neighborhood. According to initial experiments, slightly better results are obtained with a tabu list that

Table 11.3 Average relative deviations (%) for $F \| \sum w_j C_j$

m	n	MD	SA	TA	TS	GA	GD
4							
	20	0.28	0.21	0.35	0.21	0.77	0.24
	30	0.33	0.19	0.22	0.70	1.21	0.15
	40	0.64	0.24	0.30	1.09	1.51	0.23
	50	0.73	0.37	0.26	1.49	1.64	0.28
10							
	20	0.16	0.12	0.27	0.35	1.09	0.20
	30	0.69	0.50	0.74	1.51	2.03	0.32
	40	1.08	0.79	0.95	1.89	2.32	0.45
	50	0.93	0.67	0.95	1.88	2.35	0.48
Average		0.61	0.39	0.51	1.14	1.62	0.29

prevents either of the swapped jobs returning to its original position, compared with a tabu list that maintains the order between the two swapped jobs. The former list is adopted, and the tabu list length is dependent on the number of jobs.

- *GA.* The genetic algorithm employs the insertion crossover. The solutions obtained in initial experiments with the reorder crossover are inferior. A population size of 50 is used.
- *GD.* The genetic descent algorithm is a variant of GA in which a descent algorithm with the swap neighborhood is applied to each newly generated solution to find a local minimum. The reorder crossover is used, which is more effective than insertion crossover, according to initial experiments. The population size is 10.

Glass & Potts [1996] generate 10 test problems for each of the selected values of m and n. The six algorithms are each run three times on an IBM 3090 computer. Each run is initialized by randomly generating a sequence or a population, and has a time limit of 100 seconds. Average relative percentage deviations from the best known solution values are presented in Table 11.3.

The algorithms that generate the best-quality solutions are SA and GD. Although SA gives better results for the 20-job problems, GD is generally superior. This is especially noticeable for the large problems with 10 machines and 40 or 50 jobs. As in Section 8.2, GA gives the worst results. The performance of TS is rather disappointing. Algorithms MD and TA both give reasonable results, although they are inferior to both SA and GD.

8.4 Multisequencing of jobs in $J \| C_{max}$

As observed in Section 7.3, various local search algorithms are available for the job shop problem $J \| C_{max}$. Since most of the computational experiments use

the same test problems, Vaessens, Aarts & Lenstra [1996] and Balas & Vazacopoulos [1994] are able to provide a comparison between the quality of solutions generated by most of the methods. We provide similar computational results for a selection of algorithms. In addition to choosing a variety of different types of algorithms, our comparison includes all algorithms which are not dominated by others in terms of solution quality and computation time in the results given by Vaessens, Aarts & Lenstra and Balas & Vazacopoulos. The following algorithms are considered:

- *SB1.* This method is the enumerative version of the shifting bottleneck procedure of Adams, Balas & Zawack [1988], which they call SBII. The results of Vaessens, Aarts & Lenstra [1996] show that it is among the most competitive variants of this type of procedure (excluding those that incorporate guided local search).
- *SB2.* A hybrid algorithm of Balas & Vazacopoulos [1994], which they call SB-GLS2, embeds guided local search into the shifting bottleneck procedure. The guided local search uses the critical end insert neighborhood.
- *SB3.* This is another hybrid of Balas & Vazacopoulos [1994], which they call SB-RGLS1. It extends the search in SB2 by rescheduling $\lfloor \sqrt{m} \rfloor$ machines using a combination of guided local search and shifting bottleneck routines.
- *MD.* A multistart descent algorithm of Aarts et al. [1994], which the authors call MSII2, employs the extended version of the critical end transpose neighborhood, where transposes of predecessor and successor operations are attempted, as proposed by Matsuo, Suh & Sullivan [1988]. The solution quality with this algorithm is significantly better than obtained with the critical transpose neighborhood.
- *SA1.* A simulated annealing algorithm of Aarts et al. [1994] uses the critical transpose neighborhood and adopts the cooling schedule described by Van Laarhoven, Aarts & Lenstra. Algorithm SA1* is a variant of SA1 in which a much slower cooling schedule is used.
- *SA2.* A simulated annealing algorithm of Matsuo, Suh & Sullivan [1988], which the authors call CSSA, uses the extended version of the critical end transpose neighborhood, and employs an acceptance probability that is independent of objective function values.
- *SA3.* A simulated annealing algorithm of Yamada, Rosen & Nakano [1994], which the authors call CBSA, uses the critical end insert neighborhood, and employs a backtracking procedure.
- *TA.* Threshold accepting, as implemented by Aarts et al. [1994] with the critical transpose neighborhood and named TA1, uses threshold values that are multiples of those proposed by Dueck & Scheuer [1990].
- *TS1.* The tabu search algorithm of Barnes & Chambers [1995] uses the critical transpose neighborhood, and employs a backtracking procedure.
- *TS2.* The tabu search algorithm of Dell'Amico & Trubian [1993] uses a composite neighborhood consisting of generalized critical end transpose and critical end insert.

- *TS3.* The tabu search algorithm of Nowicki & Smutnicki [1996a], which the authors call TSAB, uses the critical end transpose neighborhood, and employs a backtracking procedure.
- *GS.* The guided local search procedure of Balas & Vazacopoulos [1994], which they call GLS/1, uses the critical end insert neighborhood.
- *GA1.* The genetic algorithm of Della Croce, Tadei & Volta [1995] uses a priority representation.
- *GA2.* The first genetic algorithm of Dorndorf & Pesch [1995], which the authors call P-GA, uses a heuristic set representation.
- *GA3.* The second genetic algorithm of Dorndorf & Pesch [1995], which they call SB-GA(40), is based on the shifting bottleneck procedure.
- *GA4.* The genetic algorithm of Pesch [1993], which the author calls 1MCP-GA, uses a representation that sets upper bounds for single-machine subproblems. The results of Pesch indicate that it is superior to his other genetic algorithms.
- *GD.* The genetic descent algorithm of Aarts et al. [1994], which they call GLS2, uses a descent algorithm which employs the extended version of the critical end transpose neighborhood.

For some of these algorithms, a single run is performed. However, the performance of MD, SA1, SA3, TA, TS2, and GD is averaged over five runs of the algorithm, whereas results for GA3 are averaged over two runs. Moreover, results for multistart versions of TS3 (with three starts) and GS (with four starts) are also available. A superscript indicates that the best of several runs from different starting solutions is selected. For example, $TS3^3$ refers to the best of three runs of algorithm TS3.

We compare algorithms by providing computational results from the relevant papers for a subset of the test problems of Fisher & Thompson [1963] and Lawrence [1984]. These instances are chosen to be among the 'hardest', and are therefore able to discriminate between the different algorithms. The instances considered are FT2 and FT3 of Fisher & Thompson that have $m \times n$ equal to 10×10 and 5×20 respectively, and those of Lawrence that are of type A (10×10), type B (10×15), type C (10×20), and type I (15×15).

In the computational results that are quoted in the literature, different computers are used and computation times vary from one algorithm to another. Thus, consideration should be given to both solution quality and computation time when interpreting these results. Table 11.4 shows (average) values of the objective function value C_{max}. The column OPT gives the optimal value of the maximum completion time or, when this is not known, the best known solution value is listed and marked with an asterisk. Also listed for each algorithm is the average relative deviation (ARD) of the maximum completion time from the best solution value, expressed as a percentage, where the averages are over the 10 test problems (or less for some methods because results are unavailable). Vaessens, Aarts & Lenstra [1996] are able to standardize computation times so that they are independent of the computer on which the tests are performed. Figure 11.1

Table 11.4 Average values for C_{max} for $J \| C_{max}$: best know solutions are marked with an asterisk

	OPT	SB1	SB2	SB3	MD	SA1	SA2	SA3	TA	TS1	TS2	TS3	GS	GA1	GA2	GA3	GA4	GD
FT2	930	930	930	930	1057	969	946	931	1004	930	948	930	930	965	960	938	930	982
FT3	1165	1178	–	–	1329	1216	–	–	1229	–	1167	1165	–	1199	1249	1178	1165	1294
A1	945	978	945	945	1019	977	959	–	978	947	946	946	945	989	1008	961	945	977
A4	842	860	842	842	900	855	842	–	926	848	847	842	846	–	880	863	842	859
B1	1046*	1084	1048	1048	1181	1084	1071	1055	1104	1053	1057	1055	1053	1114	1139	1074	1070	1085
B4	935	976	937	937	1076	962	973	947	1015	946	944	948	950	–	1014	960	940	981
C2	1235	1291	1240	1235	1448	1282	1274	1266	1290	1250	1252	1259	1244	–	1378	1272	1268	1301
C4	1157*	1239	1164	1164	1426	1233	1196	1193	1262	1219	1195	1164	1157	–	1336	1204	1204	1260
I1	1268	1305	1268	1268	1457	1308	1292	–	1386	1278	1289	1275	1269	1330	1373	1317	1273	1311
I3	1196*	1255	1198	1198	1442	1235	1231	1214	1323	1211	1210	1209	1213	–	1296	1251	1204	1283
ARD		3.4	0.2	0.1	14.5	3.7	2.3	1.6	7.4	1.3	1.2	0.7	0.6	4.5	8.3	2.7	1.1	5.6

ARD = Average relative deviation (%) of heuristic solution value from that of the best known solution.

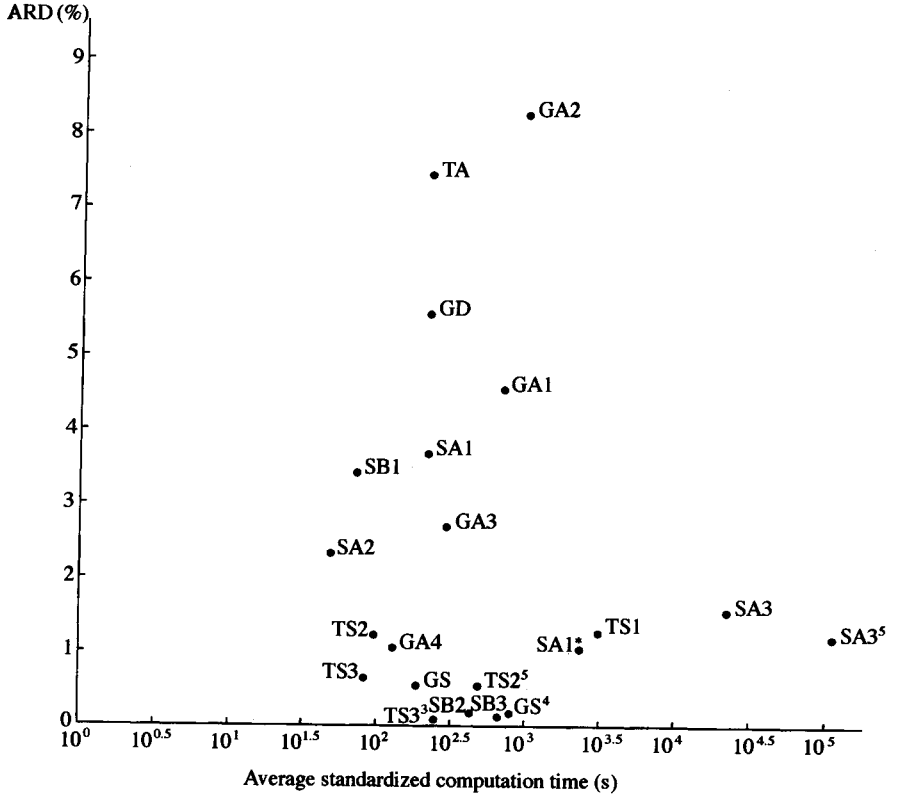

Figure 11.1 Relationship between average relative deviations (%) and standardized average computation time (in seconds)

depicts the relationship between the standardized average computation time in seconds, and the solution quality as measured by the average relative deviation ARD for the various algorithms. We omit MD from Figure 11.1 because of its large average relative deviation of 14.5%, but results for SA1*, $SA3^5$, $TS2^5$, $TS3^3$, and GS^4 are included, even though they do not appear in Table 11.4.

We first note from Table 11.4 that SB1 generates reasonable-quality solutions, whereas Figure 11.1 shows that its average computation time is modest. However, substantial improvements are obtained by including guided local search, as indicated by the results for SB2 and SB3. MD is clearly uncompetitive. Of the three simulated annealing algorithms, SA2 performs best when computation time is limited. However, comparison of SA1 and SA1* in Figure 11.1 shows that this type of algorithm can generate very good solutions if sufficient computation time is allocated. Results also show that the threshold accepting method TA is one of the least effective algorithms.

The three tabu search algorithms all produce solutions of high quality. Since the extra computational requirements of TS1 are not rewarded through better-

quality solutions, TS2 and TS3 are preferred. Moreover, Figure 11.1 shows that TS3 gives slightly better results than TS2, and $TS3^3$ generates solution values that are extremely close to the optimum. The guided local search method GS generates reasonable-quality solutions without using excessive computation time, and these solutions can be improved with a multistart version GS^4. Of the four genetic algorithms, GA4 is superior, and it is also preferred to the genetic descent algorithm GD. Taking an overview of all results, the tabu search algorithm TS3, and the multistart version $TS3^3$ are among the most competitive methods. However, high-quality solutions are also obtained with the two hybrid shifting bottleneck/guided local search methods SB2 and SB3. There is insufficient evidence at present to make a strong claim for the superiority of $TS3^3$ over SB2 and SB3.

9 CONCLUDING REMARKS

It is apparent from our discussion in the previous sections that local search techniques are the method of choice for NP-hard scheduling problems when the problem size makes a branch-and-bound approach (or other enumerative methods) impractical. Recent work on job shop problems demonstrates the power of these approaches in producing excellent solutions for intractable problems. Most studies concentrate on 'standard' scheduling models which do not contain many of the complicating features that are present in practical situations. Nevertheless, the flexibility of local search techniques enables such complications to be handled more effectively than with most other approaches.

Recent research with neighborhood search algorithms reveals some useful innovations. One successful idea is to consider a suitably chosen subset of some neighborhood, where the choice may be based on some problem-specific features. Another useful device is to allow backtracking. Using this technique, a sequence of neighborhood moves is provisionally accepted so that their effect can be explored, but if no overall improvement in solution value is observed after a specified number of iterations, the sequence of moves is rejected, and the search continues from the solution obtained prior to these exploratory moves.

The performance of genetic algorithms tends to be variable. When used as a 'black-box' technique, a genetic algorithm usually fails to generate good-quality solutions. However, by incorporating some problem-specific knowledge, which can be achieved by the use of a suitable representation of solutions, or by embedding a heuristic such as a descent method, the performance of a genetic algorithm often improves substantially.

We are not aware of any comparative computational studies for scheduling problems involving neural networks. Moreover, there is no current evidence that these methods are competitive with neighborhood search or genetic algorithms.

It is appropriate to add some comments on the empirical work that is necessary when evaluating local search algorithms. For many scheduling applications, we have found substantial difficulties in drawing firm conclusions on the relative merits of different methods. It is vital to experiment carefully with respect to

individual problem characteristics (particularly size), the computation time allowed, and the method used for generating a starting solution. There are also difficulties in knowing how much tuning of parameters is appropriate for some methods. It is remarkably easy to arrive at a faulty conclusion because of insufficient experimental work.

Finally, although there has been a large amount of research on the application of local search to scheduling problems, much remains to be done. First, the ideas we have emphasized in this chapter, such as on different representations of solutions, provide many possible avenues of research that have yet to be explored. Second, there is a wide range of different local search techniques, and it is not yet clear for which problem types the different techniques are most suited. Third, now that the potential of local search methods has been demonstrated, mainly on 'standard' problems, their use should be exploited by applying them to a broad range of more difficult practical problems.