CLASSICAL AND FLEXIBLE

JOB SHOP SCHEDULING

BY TABU SEARCH

APPROVED BY
DISSERTATION COMMITTEE:

J Wesley Barnes

Narayan Bagchi

David Morton

Valerie Taraj

Paul A. Jensen

CLASSICAL AND FLEXIBLE

JOB SHOP SCHEDULING

BY TABU SEARCH

by

John Burges Chambers, B.S., M.S.

Dissertation

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

The University of Texas at Austin

May 1996

UMI Number: 9633112

UMI
300 North Zeeb Road
Ann Arbor, MI 48103

ACKNOWLEDGEMENTS

I wish to thank those individuals who have contributed to this effort, particularly J. Wesley Barnes, for suggesting this area of investigation, for his ongoing advice and support as my dissertation advisor, and for his patience in dealing with me as a part-time doctoral student; and also Rayan Bagchi, Paul Jensen, David Morton, and Valerie Tardif, for their service as committee members and for their helpful comments.

Special appreciation is also due Clyde Poole, Coordinator of Facilities and Equipment through 1992 in the Department of Computer Sciences, The University of Texas at Austin, for allowing his staff to pursue educational objectives in parallel with their employment, and for authorizing the use of computing facilities in the department for that purpose.

CLASSICAL AND FLEXIBLE JOB SHOP SCHEDULING BY TABU SEARCH

Publication No. _____

John Burges Chambers, Ph.D.
The University of Texas at Austin, 1996

Supervisor: J. Wesley Barnes

Tabu search (TS), a local search metaheuristic, has proved a powerful strategy for solving difficult combinatorial optimization problems. This research investigates the development of an effective TS methodology for the classical job shop scheduling problem (JSSP) and the flexible job shop (FJS).

In JSSP, *n* jobs are processed to completion on *m* machines. Each job consists of *m* operations, one per machine, with known processing times and distinct technological ordering, and each machine is continuously available from time zero, processing one operation at a time without preemption. The operations are to be sequenced so as to minimize makespan.

A semidynamic TS implementation is developed from an interchange neighborhood based on a disjunctive graph representation of the underlying scheduling problem. Starting from a priority dispatching solution, iterated short term memory is used in conjunction with a historical generator method to provide effective intensification and diversification. The technique is applied to a set of standard reference problems, effective memory intervals are determined, and computational results are compared with selected established heuristics.

FJS extends JSSP by allowing an operation to be performed on more than one machine, providing a first approximation to a range of problems encountered in the planning and operation of a flexible manufacturing system. FJS is complicated by the need to determine a routing policy, *i.e.*, the assignment of operations to machines capable of performing them, in addition to the sequencing decisions.

The search neighborhood used for JSSP is reduced by accepting only certain promising candidate interchanges, and a second neighborhood of feasible routing changes is added; a move prioritization policy allows both classes of moves to be considered concurrently. Dynamic short term memory with variable bounds implements adaptive intensification, while the addition of a sliding window solution history combats cycling behavior and complements the diversification strategy. Benchmark experiments demonstrate marked improvement in the performance of single-neighborhood dynamic TS for the JSSP problem set. A small number of more difficult problems from this set are then extended by successively replicating selected machines, and computational results using multi-neighborhood dynamic TS are reviewed.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

The general problem of allocating a collection of tasks to a set of resources over time finds a wide variety of applications, including areas such as project management, manufacturing, and production, where concern for enhanced productivity and customer satisfaction motivates a desire to schedule activities in an optimal way. Unfortunately, in all but the smallest or most trivial cases, instances of such problems are remarkably difficult to solve, exhibiting a so-called *combinatorial explosion* in the number of solutions as the problem size increases. A well-known problem [Fisher and Thompson 1963] involving the scheduling of just 10 jobs on 10 machines remained open for nearly twenty-five years. Even for a problem of this apparently "small" size, and even given advances in computer technology, it is impractical to exhaustively consider all possible solutions in an effort to identify an optimum. Thus, much attention has been focused on the development of heuristic methods. This dissertation addresses the application of the tabu search metaheuristic to classical and flexible job shop scheduling.

1.1 CLASSICAL JOB SHOP SCHEDULING

The general theory of scheduling has been discussed at length by authors such as Conway, Maxwell, and Miller [1967], Baker [1974], French [1982], and Lawler *et al.* [1993]. Additionally, MacCarthy and Liu [1993a] have surveyed the state of the art in scheduling research and suggested ways in which scheduling researchers may better address the needs of scheduling practice.

1.1.1 Problem formulation

Terminology and notation

In a general sense, the scheduling problem addresses the allocation of some set of tasks to some set of resources capable of performing those tasks. The goal is to optimize some (typically time-dependent) measure of goodness while respecting feasibility constraints appropriate to the problem definition. Because job shop scheduling finds perhaps its most natural expression in the context of manufacturing and production, the terminology most often adopted reflects that environment.

Consider a set of N **operations** which are to be processed on a set of m **machines**. It may be the case that an operation can be processed on any available machine, or only on a certain subset of machines, or perhaps only on one particular machine. Typically, the operations are related by conceptual considerations (*e.g.*, operations related to the processing of one particular part) or by **technological constraints** (*e.g.*, a part should be milled, deburred, and polished in that order). Based on such relationships, the set of operations is naturally partitioned into n exhaustive, disjoint sets or **jobs**.

We denote the nonnegative **processing time** of operation i in job j on machine k by p_{ijk}. (If, for each machine k, only one operation in job j requires processing on that machine, the operation index i may be omitted; the meaning will usually be clear from the context.) A job may have a **release date**, r_j, a time before which processing may not begin, and a **due date**, d_j, a time by which all processing must be completed (some penalty for tardiness being implicit). The **completion time**, C_j, is the time at which the processing of job j is finished, and includes any **waiting time** spent by operations in job j while needed machines finish with other tasks. (The waiting time should not be confused with the **idle time** spent by a machine while it waits for work to be made available to it for processing.)

A **performance measure** is simply a cost or objective function which measures the relative goodness of a schedule. In the presence of due dates, for example, it may be desirable to minimize the average tardiness or total number of late jobs in order to avoid disappointing a customer. A **regular performance measure** is a performance measure (to be minimized) which is nondecreasing as a function of the job completion times. One such regular performance measure, and the one with which we will concern ourselves in Chapters 3 and 4, is the **makespan**, C_{max}, the maximum completion time over all jobs (that is, the time at which all processing has been completed). A minimum makespan schedule in some sense reflects a superior utilization of machine resources; for example, if the length of a single production run can be reduced, the number of runs which can be executed during a given production period, and the resulting production volume, can be increased.

Assumptions for the job shop

Certain assumptions are useful in modeling the classical job shop scheduling problem (see, *e.g.*, [Baker 1974] or [French 1982]).

1. There is only one of each type of machine. A machine can process only one operation at a time. An operation, once begun, is processed to completion without interruption (no **preemption**). All machines are continuously available, without breakdown, from time 0.
2. Each job consists of m operations, one per machine type; that is, each job requires processing on each machine exactly one time (no **reentrant jobs**). Every job is available from time 0 ($r_j = 0 \; \forall \; j$), and is processed to completion (no **cancellation**). For the minimum makespan problem, we need not consider due dates.

3. All processing times are strictly positive and are determined in advance. Also, processing times are independent of schedule; if **setup times** or **changeover times** are required, they are included in the processing times.
4. Technological constraints are determined in advance.

The technological constraints specify the order in which operations in a job must be performed. More precisely, they specify the order in which a job must visit the machines that will perform those operations. This machine order associated with a job is typically referred to as the job's **routing**. (The reader should not confuse technological constraints with **precedence constraints** specifying an ordering of operations belonging to different jobs.)

A routing may be **complete**, specifying the exact ordering of all the operations in a job, or **partial**, leaving undetermined the ordering of certain operations relative to others (see, *e.g.*, Russo [1965]). A job shop problem for which no routing is given for any job is known as an **open shop** problem. If each job has a complete routing specified, and the routing is identical for every job, the problem is known as a **flow shop** problem. (One may also see reference made to a **permutation flow shop**, a flow shop in which the jobs on each machine must be performed in the same sequence.) Finally, we will refer to a problem which satisfies assumptions (1) - (4), and for which each job has a unique, complete routing, as a **classical job shop** or **general job shop** problem.

With regard to assumption (3), it is sometimes convenient to model the special case in which a job does not require processing on a particular machine by setting the job's processing time on that machine to zero, yielding a "dummy" operation. However, the question then remains of when to schedule such an operation, since the next operation in the job could conceivably be delayed while the dummy operation awaits release of a

machine it does not actually require. Hefetz and Adiri [1982], in particular, have addressed the computational impact of this modeling practice.

Often, this representation motivates a computer implementation using static data arrays to record the assignment of operations to machines. In such an implementation, a zero-length operation representing a missing operation serves merely as a "placeholder". The implementor can avoid this situation entirely by the use of more sophisticated data structures (*e.g.*, linked lists) which allow a missing operation to be, in fact, missing. Such approaches also allow the implementor far more flexibility in representing more general (and more realistic) problems with incomplete routings or reentrant jobs.

1.1.2 Types of schedules

Table 1.1 gives data for a sample 3-job, 3-machine classical job shop problem. For example, job 2 is processed on machines 2, 3, and 1, in that order, with respective processing times of 4, 1, and 3 (the time units are assumed consistent but otherwise unspecified). Figure 1.1 shows **Gantt charts** illustrating several schedules which might be constructed for this problem.

Figure 1.1(a) is a **semiactive schedule**, with a makespan of 20. It is characterized by the fact that the operations on each machine are shifted as

Table 1.1

Sample 3-job, 3-machine problem

j	p_{j1}	p_{j2}	p_{j3}	routing
1	7	1	1	1 2 3
2	3	4	1	2 3 1
3	4	2	3	3 1 2

far to the left (*i.e.*, earlier in time) as possible *without changing the sequence of operations on that machine or yielding an infeasible schedule*. It is worthwhile to note that, in fact, an infinite number of *schedules* can be constructed from one given *sequence* by inserting arbitrary amounts of idle time between operations. However, it is not difficult to show that, for a given sequence, and with respect to a regular measure of performance, the corresponding semiactive schedule is no worse than any other which can be constructed from that sequence. Hence, the search for a solution can be restricted to the finite set of semiactive schedules corresponding to the finite set of feasible sequences (implying that an optimal solution must exist). The terms "sequence" and "schedule" are typically used interchangeably.

Figure 1.1(b) is an **active schedule**, with a makespan of 17. This schedule has a better makespan than that in Figure 1.1(a) because the sequence on machine 3 has been rearranged to allow the operation from job 2 to be processed during an otherwise extended idle period, and without delaying the processing of the operation from job 1. In an active schedule, no operation can be moved earlier *without delaying another operation or yielding an infeasible schedule*. Again, it is not difficult to argue that the set of active schedules, a subset of the semiactive schedules, is dominant with respect to a regular measure of performance.

Figure 1.1(c) shows a **nondelay schedule**, with a makespan of 16. In a nondelay schedule, *no machine is kept idle if it could be processing some operation*. Although the set of nondelay schedules is a subset of the set of active schedules, an optimal nondelay schedule does not necessarily exist.

Finally, Figure 1.1(d) shows a schedule with a makespan of 14, which happens to be optimal for the sample problem. The proof that a schedule is optimal typically must depend on exhaustive enumeration (impractical for problems of realistic size), or on the existence of lower bounds (addressed later in this chapter). (The reader may note that, for the sample problem, a

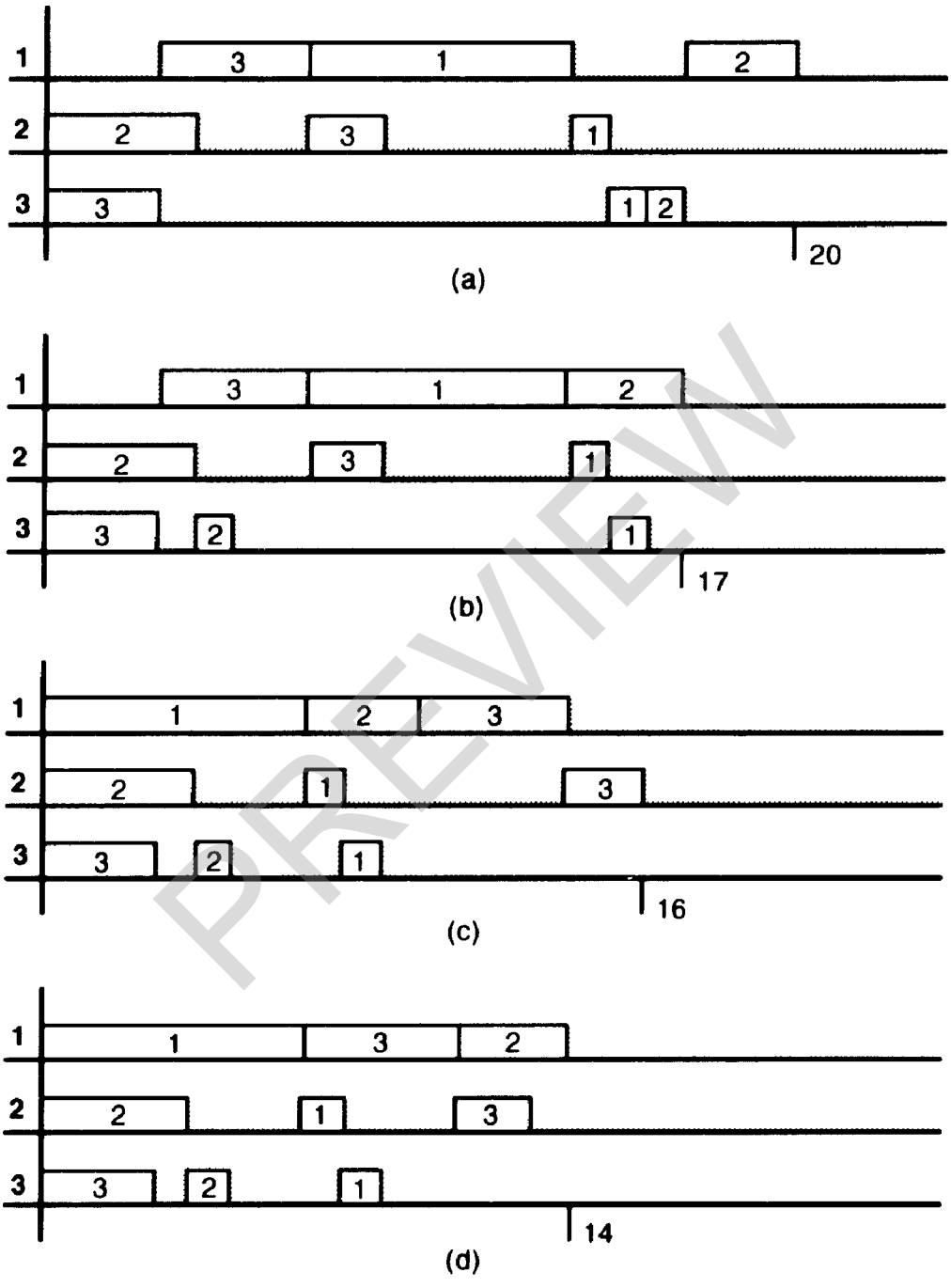

Figure 1.1 Types of schedules

useful, if trivial, lower bound exists which is the maximum cumulative processing time required by any job or machine. For the example, this bound occurs for machine 1, which is needed for a total of 14 time units. Therefore, the schedule with makespan of 14 must be optimal.)

1.1.3 An alternative formulation

Many researchers in job shop scheduling have relied on the **disjunctive graph** representation of the problem [Roy and Sussman 1964]. In this representation, each operation is portrayed as a node in a graph. Conjunctive arcs, of fixed orientation, correspond to technological constraints on the order of operations in a given job. Disjunctive arcs, of assignable orientation, represent the sequencing order of operations on a given machine. Dummy nodes are typically included as the source (sink) for operations with no job predecessors (successors). A **selection** assigns an orientation to each disjunctive arc, establishing a particular schedule. The processing time of an operation may be viewed as the length of each post-selection out-arc of the node representing that operation. The makespan of the resulting schedule is then the length of a longest path in the graph from source to sink. It is well known that the graph representing a feasible schedule must necessarily be free of cycles.

Figure 1.2(a) shows a disjunctive graph, before selection, representing the sample problem of Table 1.1. Node (j,k) represents the operation corresponding to the processing of job j on machine k. Node s (t) is the source (sink) for operations without job predecessors (successors). Figure 1.2(b) shows a selection corresponding to the optimal schedule of Figure 1.1(d). Note that arcs providing redundant sequencing information for a given machine, *e.g.*, ((1,1),(2,1)), are not drawn. Such arcs are unnecessary in an efficient computation of makespan [Adams, Balas, and Zawack 1988].

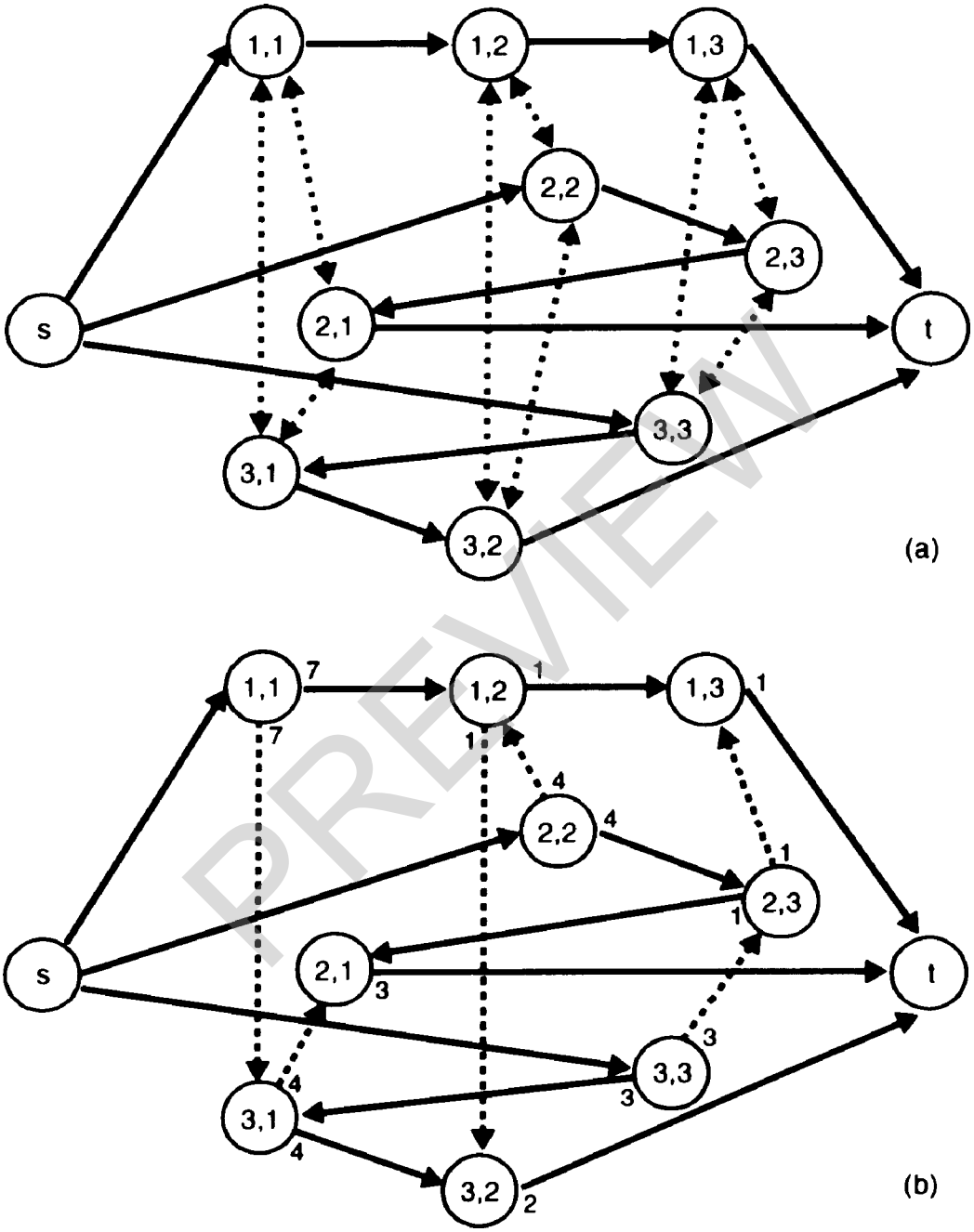

Figure 1.2 Disjunctive graph formulation

Although it is possible to work from the disjunctive graph representation directly, the greater value lies perhaps in the *properties* of such graphs [Balas 1969; Balas 1970; van Laarhoven 1988; van Laarhoven, Aarts, and Lenstra 1992], which yield insight into the formulation of solution methodologies, and which are explored further in Chapters 2 and 3.

1.1.4 Solution methodologies

Complexity

As noted above, the search for an optimal solution to a classical job shop problem involves, at most, an examination of the finite set of semiactive schedules corresponding to the set of feasible sequences, itself a subset of the set of all possible sequences. Unfortunately, for an n-job, m-machine problem, there are $(n!)^m$ possible sequences, sufficiently many of which will be feasible to suggest that solution by exhaustive enumeration is impractical, within any reasonable time constraint, for problems of even modest size. By way of example, the notorious 10x10 instance of Fisher and Thompson [1963] remained unsolved for a quarter century, yielding finally to a bounded heuristic procedure requiring some 5 hours of CPU time [Carlier and Pinson 1989].

As it turns out, many scheduling problems are computationally intractable. The complexity of various scheduling problems has been reviewed by numerous authors, including Rinooy Kan [1976], MacCarthy and Liu [1993a], Lawler *et al.* [1993], and Vaessens, Aarts, and Lenstra [1994]. Palem [1986] explores the complexity of scheduling in the presence of precedence constraints, and Hefetz and Adiri [1982] consider the computational impact of missing operations. In particular, the general job shop scheduling problem, except for certain very restricted cases, is strongly NP-hard [Garey, Johnson, and Sethi 1976; Rinooy Kan 1976], and therefore most approaches employ a heuristic strategy. (For a general discussion of computational complexity, see

Garey and Johnson [1979], Papadimitriou and Steiglitz [1982], or French [1982].)

Heuristic strategies

One-pass constructive methods relying on priority dispatching rules have been used with varying degrees of success. Baker [1974] describes one approach to generating active and nondelay schedules using a dispatching adaptation of an enumerative heuristic proposed by Giffler and Thompson [1960]. Such a procedure constructs a schedule from the left in *N* steps, at each step scheduling an operation selected (based on the type of schedule desired) from among the set of currently schedulable operations. If more than one operation could be selected, a choice is made based on *priority rules* which evaluate quantities such as Shortest Processing Time or Most Work Remaining. The effectiveness of numerous priority rules has been discussed by Jeremiah, Lalchandani, and Schrage [1964], Conway, Maxwell, and Miller [1967], and Panwalkar and Iskander [1977], with newer investigations being reported by authors such as Vepsalainen and Morton [1987]. The strength of priority dispatching lies in its ability to rapidly generate reasonably good schedules in a well-understood fashion; these schedules may then be used as upper bounds or starting solutions for other techniques. (See also Chapters 3 and 4.)

Traditional integer programming methods, such as Greenberg's [1968] mixed integer programming approach, and the surrogate constraint relaxations employed by Fisher *et al.* [1983], have proved less satisfactory in the area of scheduling than might be expected, but related research has yielded lower bounds useful in the development of other heuristics. For example, Lageweg, Lenstra, and Rinooy Kan [1977] note that many effective lower bounds for the job shop are simply special cases of the single machine bound, and Applegate and Cook [1991] review eight valid inequalities and

develop a cutting plane method for establishing lower bounds in conjunction with an enumerative heuristic.

In addition to the enumerative schemes proposed by Giffler and Thompson [1960] and by Applegate and Cook [1991], branch and bound methods of various sorts have been described by numerous authors, including Balas [1969], Ashour and Hiremath [1973], Lageweg, Lenstra, and Rinooy Kan [1977], Adams, Balas, and Zawack [1988], Carlier and Pinson [1989], Balas, Lenstra, and Vazacopoulos [1992], and Dauzere-Peres and Lasserre [1993]. Most of these schemes have yielded good results, albeit with significant effort invested in the proof of optimality. For example, Carlier and Pinson [1989] offered the first solution to the Fisher-Thompson 10x10 instance [1963], requiring 3305 seconds to locate the optimum, but an additional 14680 seconds to prove optimality (CPU times quoted for a Prime 2655 computer).

Local search techniques

Several effective solution methodologies based on local search have been developed for job shop scheduling. Local search considers a neighborhood structure on a solution space and, starting from an initial feasible solution, iteratively moves to successively better solutions until no further improvement is possible. (A simple sketch of the procedure is given in Figure 1.3.) A general discussion of local search may be found in Papadimitriou and Steiglitz [1982] or Glover and Greenberg [1989], while Aarts *et al.* [1994] and Vaessens, Aarts, and Lenstra [1994] review the applicability of the method to the job shop problem. Although generalized search procedures can probably never be competitive with "tuned" heuristics which can incorporate detailed knowledge about the cost function in the search function [Wolpert and Macready 1995], they are nevertheless powerful tools which in practice can yield high quality solutions with modest computational effort.

```
// Given:
//   N() a neighborhood function, C() a cost function,
//   S^0 an initial feasible solution, Z^0 = C(S^0), and
//   X^0 = { x | x ∈ N(S^0) }
S = S^0; Z = Z^0; X = X^0;

local search {
    while ( X != {∅} ) {
        select a feasible S' ∈ X;
        Z' = C(S');
        if ( Z' < Z ) {
            S = S';
            Z = Z';
            X = { x | x ∈ N(S') };
        }
        else X = X - {S'};
    }
    return S;
}
```

Figure 1.3 General local search

A critical component of any local search strategy is the selection of an appropriate and effective neighborhood. A number of neighborhoods have been proposed for the job shop problem by Balas [1969], Suh [1988] and Matsuo, Suh, and Sullivan [1988], Dell'Amico and Trubian [1993], Nowicki and Smutnicki [1993], and others. Many of these are reviewed by Vaessens, Aarts, and Lenstra [1994]. Some of these neighborhoods exhibit properties of so-called *combinatorial leverage*, offering the advantage of exponential search space with polynomial effort [Glover 1991a].

An unfortunate failing of general local search is its tendency to become trapped at a local minimum which is "close" to the initial solution, but which may be "far" from the global optimum. Various adaptations have been developed to allow the search to explore other regions of the solution space; several of these are described by Glover and Greenberg [1989] and by