Guided Local Search with Shifting Bottleneck for Job Shop Scheduling

Egon Balas • Alkis Vazacopoulos
Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213-3890
Fairleigh Dickinson University, Madison, New Jersey 07940

Many recently developed local search procedures for job shop scheduling use interchange of operations, embedded in a simulated annealing or tabu search framework. We develop a new variable depth search procedure, GLS (Guided Local Search), based on an interchange scheme and using the new concept of neighborhood trees. Structural properties of the neighborhood are used to guide the search in promising directions. While this procedure competes successfully with others even as a stand-alone, a hybrid procedure that embeds GLS into a Shifting Bottleneck framework and takes advantage of the differences between the two neighborhood structures proves to be particularly efficient. We report extensive computational testing on all the problems available from the literature.
(Job Shop Scheduling; Local Search; Shifting Bottleneck; Neighborhood Trees)

1. Introduction

In the job shop scheduling problem, jobs are to be processed on machines with the objective of minimizing some function of the completion times of the jobs, subject to the constraints that: (i) the sequence of machines for each job is prescribed, and (ii) each machine can process at most one job at a time. The processing of a job on a machine is called an operation, and its duration is a given constant. The objective chosen here is to minimize the time needed to complete all the jobs, called the makespan.

Let $N = \{0, 1, \ldots, n\}$ denote the set of operations, with 0 and n the dummy "start" and "finish" operations, respectively, M the set of machines, A the set of pairs of operations constrained by precedence relations representing condition (i) above, and E_k the set of pairs of operations to be performed on machine k and that therefore cannot overlap in time, as specified in (ii). Further, let $p_j > 0$ denote the (fixed) duration or processing time, and t_j the (variable) start time of operation j. The problem can then be stated as:

$$\text{minimize } t_n,$$

subject to

$$t_j - t_i \geq p_i, \quad (i, j) \in A,$$

$$t_i \geq 0, \quad i \in N,$$

$$t_j - t_i \geq p_i \vee t_i - t_j \geq p_j, \quad (i, j) \in E_k, k \in M. \quad (P)$$

Any feasible solution to (P) is called a schedule.

For literature on job shop scheduling, see Conway et al. (1967), Baker (1974), Lenstra (1976), Rinnooy Kan (1976), French (1982), Blazewicz et al. (1992), Morton and Pentico (1993), and their references.

It is useful to represent this problem on a *disjunctive graph* $G := (N, A, E)$ with node set N, (directed) arc set A, and (undirected, but orientable) edge set E. Alternatively, each edge of E can be viewed as a pair of oppositely directed arcs. The length of an arc $(i, j) \in A$ is p_i, whereas the length of an edge $(i, j) \in E$ is either p_i or p_j, depending on its orientation. Each machine k corresponds to a set N_k of nodes (operations) and a set E_k of edges that together form a disjunctive clique.

$D = (N, A)$ denotes the directed graph obtained from G by removing all disjunctive arcs. A selection S_k in E_k contains exactly one member of each disjunctive arc pair of E_k. A selection is acyclic if it contains no directed cycle. Each acyclic selection S_k corresponds to a unique

0025-1909/98/4402/0262$05.00

sequence of the operations pertaining to machine k, and vice versa. Therefore, sequencing machine k means choosing an acyclic selection in E_k.

A complete selection S consists of the union of selections S_k, one in each E_k, $k \in M$. Picking a complete selection S, i.e., replacing the disjunctive arc set E with the ordinary (conjunctive) arc set S, gives rise to the (standard) directed graph $D_S = (N, A \cup S)$. A complete selection is acyclic if the digraph D_S is acyclic. Every acyclic complete selection S defines a family of schedules, and every schedule belongs to exactly one such family. Furthermore, if $L(u, v)$ denotes the length of a longest path from u to v in D_S, and if $\hat{t}$ is defined by $\hat{t}_u = L(0, u)$ for all $u \in N$, then $\hat{t}$ is an optimal schedule for the selection S; and hence the makespan $\hat{t}_n$ of the schedule $\hat{t}$ is equal to the length of a longest (critical) path in D_S. Thus, in the language of disjunctive graphs, our problem is that of finding an acyclic complete selection $S \subset E$ that minimizes the length of a longest path in the directed graph D_S.

Job shop scheduling is known to be a difficult, strongly NP-complete problem. In fact, practical experience shows that it is among the hardest combinatorial optimization problems. This is dramatically illustrated by the fact that an instance involving 10 jobs and 10 machines, proposed by Fisher and Thompson (1963), remained unsolved for more than a quarter of a century, even though every available algorithm has been tried on it. Optimization algorithms for job shop scheduling usually proceed by branch and bound; and among the more recent and successful ones are those of Carlier and Pinson (1989) and Applegate and Cook (1991).

Approximation procedures, or heuristics, were first developed on the basis of dispatching rules, or priority rules for choosing the next job to be "dispatched" (scheduled) in the process of generating a so called active schedule. These procedures are very fast, but the quality of the solutions that they produce usually leaves plenty of room for improvement. A more elaborate approach, which at a higher computational cost tends to produce considerably better approximations, is the Shifting Bottleneck Procedure of Adams et al. (1988), based on repeatedly optimizing the sequence on each individual machine, while keeping the sequence on all other machines fixed. If pursued until no more improvements can be obtained, this procedure yields a local optimum over the neighborhood defined by all those schedules obtainable from a given one by changing the sequence on any single machine.

More recently, some local search procedures were developed that use a neighborhood defined by reversing some disjunctive arc (u, v) on a critical path in the graph D_S associated with a given selection S. These procedures, to be briefly described in the next section, are known as interchange heuristics, since reversing the arc (u, v) amounts to interchanging the positions of nodes u and v in the sequence defined by the critical path. In this paper we use an interchange scheme, called Guided Local Search (GLS), based on reversing more than one disjunctive arc at a time. Our procedure thus investigates a considerably larger neighborhood than the heuristics based on pairwise interchange. Also, this neighborhood is defined in a way that allows for interchanges involving sets of arcs of varying sizes, so in this sense the search is of variable depth. Further properties of the neighborhood are used to guide the search into promising directions. Finally, while the earlier search procedures use simulated annealing or tabu search as a framework enabling them to overcome local optima, we use for this purpose the new concept of neighborhood trees.

The next section describes the neighborhood structure that forms the basis of our search procedure. Its central concepts are forward and backward interchanges, and critical pairs of operations that are candidates for such interchanges. Section 3 discusses the evaluations and guideposts that direct our search. These are based on further structural properties of the neighborhood. In particular, interchanges that increase the makespan are shown to have special properties that can be exploited in the search. Section 4 states the GLS procedure, centered around the concept of neighborhood trees. Section 5 discusses the embedding of GLS into the Shifting Bottleneck Procedure. The combination of these two search procedures based on very different neighborhood definitions turns out to be a great success, as demonstrated by the computational results discussed in §6.

2. The Neighborhood Structure

Local search procedures explore a certain neighborhood of a given complete selection S and associated schedule

$t(S)$. The neighborhood can be defined as the set of complete selections (and associated schedules) obtainable from S by some specified perturbation. The perturbation that we will consider takes a pair of operations u, v to be performed on the same machine, such that u and v are both on the same critical path $P(0, n)$ in D_S, and moves u *right after* v, or v *right before* u. When u and v are adjacent in $P(0, n)$, the two kinds of perturbation are the same, and they amount to interchanging u with v, i.e., replacing the arc (u, v) with its complement (v, u). In the general case, however, the two kinds of perturbation differ and each of them involves the replacement of several arcs with their complements. We will call the above perturbation an *interchange on u and v*: a *forward interchange* if u is moved right after v, a *backward interchange* if v is moved right before u.

Moves of this type, sometimes called forward shift and backward shift, have been used earlier in the context of branch and bound procedures for one-machine scheduling (see Carlier 1982 or the block approach of Grabowski et al. 1986), and very recently also in the context of heuristic search (see Dell'Amico and Trubian 1993). We use the term interchange rather than shift, to stress the connection with the interchange heuristics that served as our immediate inspiration. Indeed, our neighborhood concept is a direct extension of the earlier one defined by the interchange of a pair of operations u and v to be performed on the same machine and *adjacent* on $P(0, n)$. The neighborhood structure defined in this narrower way has been used by several authors in the last few years. An iterative improvement procedure based on this neighborhood was developed and tested by Laarhoven et al. (1992). At first, the results were unsatisfactory; but when the procedure was embedded into a simulated annealing framework, i.e., when makespan-increasing interchanges were accepted with a nonzero probability, the results improved considerably. Even better results, with shorter computing times, were obtained by Matsuo et al. (1988) by simulated annealing based on interchanges restricted to pairs u, v adjacent on $P(0, n)$, such that either the job-predecessor of u, or the job-successor of v, also belongs to $P(0, n)$.

More recently, Nowicki and Smutnicki (1993) embedded the interchange of operations adjacent on $P(0, n)$ into a tabu search framework, another approach that accepts makespan-increasing moves. Their computational results compare favorably with those obtained by simulated annealing.

All of the above procedures restrict the interchange to pairs u, v such that the arc (u, v) is on $P(0, n)$. As observed in Balas (1969), unless this condition holds, the interchange of u and v cannot reduce the makespan; but if the condition holds, then the interchange yields a complete selection that is acyclic. Another important observation, due to Matsuo et al. (1988), is that unless the job-predecessor of u or the job-successor of v is on the critical path $P(0, n)$ containing u and v, the interchange cannot reduce the makespan.

Next we give a detailed description of the neighborhood structure considered in this paper.

For any operation u, we will denote by $\alpha(u)$ and $\gamma(u)$ the job-predecessor and job-successor, respectively, of u; i.e., $(\alpha(u), u)$ and $(u, \gamma(u))$ are arcs of the conjunctive graph D. Furthermore, given a complete selection S and associated schedule $t(S)$, for any operation u we will denote by $\beta(u)$ and $\delta(u)$ the machine-predecessor and machine-successor (if it exists), respectively, of u; i.e., the operation (if any) that precedes u, and succeeds u, respectively, on the machine performing operation u. In other words, $(\beta(u), u)$ and $(u, \delta(u))$ (if they exist) are arcs of S, more precisely of S_k, where k is the machine performing operation u.

The proposition below follows from results by Potts (1980), and was used in a branch and bound context by Carlier (1982) and Grabowski et al. (1986). We state it without proof. Let $P(0, n)$ be a critical path containing u and v.

PROPOSITION 2.1. *If neither $\alpha(u)$, nor $\gamma(v)$ is contained in $P(0, n)$, then an interchange on u, v cannot reduce the makespan.*

Thus for an interchange on u, v to reduce the makespan, it is necessary that a critical path $P(0, n)$ containing u and v also contain at least one of the nodes $\alpha(u)$ and $\gamma(v)$.

Next we explore conditions under which an interchange on u and v is guaranteed not to create a cycle.

PROPOSITION 2.2. *If a critical path $P(0, n)$ containing u and v also contains $\gamma(v)$, and*

$$L(v, n) \geq L(\gamma(u), n), \qquad (2.1)$$

then a forward interchange on u and v yields an acyclic complete selection.

PROOF. By contradiction: suppose moving u right after v creates a cycle C. Then C contains either $(u, \gamma(u))$, or $(u, \delta(v))$. If $(u, \gamma(u)) \in C$, there is a path in D_S from $\gamma(u)$ to v, and hence $L(\gamma(u), n) > L(v, n)$, contrary to (2.1). If $(u, \delta(v)) \in C$, there is a path in D_S from $\delta(v)$ to v, contrary to the assumption that D_S is acyclic. □

By analogy, we have Proposition 2.3.

PROPOSITION 2.3. *If a critical path $P(0, n)$ containing u and v also contains $\alpha(u)$, and*

$$L(0, u) + p_u \geq L(0, \alpha(v)) + p_{\alpha(v)}, \qquad (2.2)$$

then a backward interchange on u and v yields an acyclic complete selection.

PROOF. Parallels that of Proposition 2.2. □

Notice that if u and v are adjacent in $P(0, n)$, then conditions (2.1) and (2.2) are both satisfied even when $P(0, n)$ contains neither $\alpha(u)$ nor $\gamma(v)$. Indeed, in this case $u = \beta(v)$ and $v = \delta(u)$. Thus (2.1) says that the longest path from u to n is through the machine-successor of u rather than its job-successor (which is why (u, v) is on $P(0, n)$); and (2.2) says that the longest path from 0 to v is through the machine-predecessor of v rather than its job-predecessor.

Two operations u, v to be performed on the same machine will be said to form a *critical* pair if they lie on a critical path $P(0, n)$ in D_S such that all the operations in $P(u, v)$ pertain to the same machine, and either $P(0, n)$ contains $\gamma(v)$ and (2.1) holds, or $P(0, n)$ contains $\alpha(u)$ and (2.2) holds. In the first case we will say that the critical pair u, v is a *forward pair*, i.e., a candidate for a forward interchange, while in the second we will say that u, v is a *backward pair*, i.e., a candidate for a backward interchange.

Our search neighborhood can now be concisely defined as the set of complete selections (and associated schedules) obtainable from S *by performing an interchange on a critical pair u, v.*

The next two propositions describe some basic properties of interchanges on a critical pair.

PROPOSITION 2.4. *A forward interchange on a critical pair u, v does not affect the longest paths from $\gamma(u)$ to n, from $\gamma(v)$ to n, and from $\delta(v)$ to n.*

PROOF. The forward interchange replaces the arcs $(u, l_1), \ldots, (u, l_k), (u, v)$ with their complements. For $\mu = \gamma(u), \gamma(v)$ and $\delta(v)$, let $P(\mu, n)$ and $P'(\mu, n)$ be longest paths from μ to n in D_S and in $D_{S^{u,v}}$, i.e. before and after the interchange, respectively. Clearly, $P(\mu, n)$ cannot contain any of the arcs $(u, l_1), \ldots, (u, l_k), (u, v)$, for any of the three possible values of μ. Hence the removal of any of these arcs from D_S leaves intact $P(\mu, n)$, which remains a path in $D_{S^{u,v}}$. The only way the status of $P(\mu, n)$ can be affected by the interchange, is for the introduction of the arcs $(l_1, u), \ldots, (l_k, u), (v, u)$ into $D_{S^{u,v}}$ to create a new longest path $P'(\mu, n)$ from μ to n, longer than $P(\mu, n)$. For this to happen, $P'(\mu, n)$ would have to contain one of the arcs $(l_1, u), \ldots, (l_k, u), (v, u)$. Suppose this is the case.

Let $\mu = \gamma(u)$. Then $D_{S^{u,v}}$ contains a path from $\gamma(u)$ to u, and of course the arc $(u, \gamma(u))$, i.e., $D_{S^{u,v}}$ is not acyclic. But this contradicts Proposition 2.2.

Now let $\mu = \gamma(v)$. Since $P'(\gamma(v), n)$ cannot contain (v, u), it must contain (l_i, u) for some $i \in \{1, \ldots, k\}$. But if $D_{S^{u,v}}$ contains a path from $\gamma(v)$ to l_i not involving u, Then D_S must have contained the same path, since all the arcs removed from D_S by the interchange involve u. But D_S also contained a path from l_i to v and hence to $\gamma(v)$, contrary to the assumption that D_S was acyclic.

Finally, let $\mu = \delta(v)$. Then u is on a longest path $P'(\delta(v), n)$; but at the same time $\delta(v)$, the machine successor of v before the interchange, becomes the machine successor of u after the interchange; i.e. u both precedes and succedes $\delta(v)$ in $D_{S^{u,v}}$, a contradiction. □

PROPOSITION 2.5. *A backward interchange on a critical pair u, v does not affect the longest paths from 0 to $\alpha(v)$, from 0 to $\alpha(u)$, and from 0 to $\beta(u)$.*

PROOF. Parallels that of Proposition 2.4. □

Note that the depth of this neighborhood, when measured by the number of disjunctive arcs that can get reversed without leaving the neighborhood, is variable. Whereas the interchange of two operations u, v *adjacent* on a critical path always involves the reversal of a single arc (u, v), i.e., the replacement of (u, v) with its complement (v, u), the more general interchange introduced here involves the reversal of a potentially large number of arcs. Indeed, let $(u, l_1, \ldots, l_k, v, \gamma(v))$ be the segment of a critical path containing u, v and $\gamma(v)$. Then a forward interchange on u and v

involves the reversal of the arcs $(u, l_1), \ldots, (u, l_k), (u, v)$. Similarly, if $(\alpha(u), u, l_1, \ldots, l_k, v)$ is the segment of a critical path containing u, v and $\alpha(u)$, then a backward interchange on u and v requires the reversal of the arcs $(u, v), (l_1, v), \ldots, (l_k, v)$.

Thus our procedure is a particular kind of variable-depth interchange.

On the other hand, an interchange on u, v as defined above is not the same as the interchange *of* u and v. Interchanging u and v would mean replacing u with v and v with u in the sequence of operations, a move equivalent to performing a forward interchange on u, v, and then a backward interchange on l_1, v. Such a move would involve the reversal of all arcs $(u, l_1), \ldots, (u, l_k), (u, v)$ and $(l_1, v), \ldots, (l_k, v)$.

Among the neighborhoods used in other local search procedures, the one that is closest to ours is that of Dell'Amico and Trubian (1993). Actually, the neighborhood used therein is slightly larger than ours, as it allows for an interchange on u and v (a shift) even when the critical path containing u and v contains neither $\alpha(u)$ nor $\delta(v)$.

3. Evaluations and Guideposts

We call evaluations our estimates of the makespan resulting from an interchange on a critical pair. These evaluations are used intensely throughout the procedure for ranking (prioritizing) the candidates for an interchange at every step. The makespan resulting from an interchange can of course be calculated exactly in $O(n)$ time; however, doing this for every candidate at every step becomes expensive, and we have opted for a less costly approximation procedure.

We will denote by $L(i, j)$ and $L^{u,v}(i, j)$ the length of a longest path from i to j (if it exists) before and after an interchange on u, v, respectively, and by $\lambda^{u,v}(i, j)$ our evaluation (estimate) of $L^{u,v}(i, j)$, namely

$$\lambda^{u,v}(0, n) = \max\{\lambda^{u,v}(0, w) + \lambda^{u,v}(w, n) : w \in Q\},$$

where $Q := \{u, l_1, \ldots, l_k, v\}$ is the segment of the critical path $P(0, n)$ containing u and v. Here the estimates $\lambda^{u,v}(0, w)$ and $\lambda^{u,v}(w, n)$ are calculated recursively as follows.

Case 1. The interchange on u, v is a forward one. Then we have

$$\lambda^{u,v}(0, l_1) = \begin{cases} L(0, \alpha(l_1)) + p_{\alpha(l_1)} \\ \quad \text{if } u \text{ was the first operation on its machine,} \\ \max\{L(0, \alpha(l_1)) + p_{\alpha(l_1)}, \\ \quad L(0, \beta(u)) + p_{\beta(u)}\} \text{ otherwise;} \end{cases}$$

for $w \in \{l_2, \ldots, l_k, v\}$,

$$\lambda^{u,v}(0, w) = \max\{L(0, \alpha(w)) + p_{\alpha(w)}, \lambda^{u,v}(0, \beta(w)) + p_{\beta(w)}\},$$

and

$$\lambda^{u,v}(0, u) = \max\{L(0, \alpha(u)) + p_{\alpha(u)}, \lambda^{u,v}(0, v) + p_v\}.$$

Further, we have

$$\lambda^{u,v}(u, n) = \begin{cases} p_u + L(\gamma(u), n) \\ \quad \text{if } v \text{ was the last operation on its machine,} \\ p_u + \max\{L(\gamma(u, n), L(\delta(v), n)\} \\ \quad \text{otherwise,} \end{cases}$$

$$\lambda^{u,v}(v, n) = p_v + \max\{L(\gamma(v), n), \lambda^{u,v}(u, n)\},$$

and for $w \in \{l_1, \ldots, l_k\}$,

$$\lambda^{u,v}(w, n) = p_w + \max\{L(\gamma(w), n), \lambda^{u,v}(\delta(w), n)\}.$$

Case 2. The interchange on u, v is a backward one. Then we have

$$\lambda^{u,v}(0, v) = \begin{cases} L(0, \alpha(v)) + p_{\alpha(v)} \\ \quad \text{if } u \text{ as the first operation on its machine,} \\ \max\{L(0, \alpha(v)) + p_{\alpha(v)}, L(0, \beta(u)) + p_{\beta(u)}\} \\ \quad \text{otherwise,} \end{cases}$$

$$\lambda^{u,v}(0, u) = \max\{L(0, \alpha(u)) + p_{\alpha(u)}, \lambda^{u,v}(0, v) + p_v\},$$

and for $w \in \{l_1, \ldots, l_k\}$,

$$\lambda^{u,v}(0, w) = \max\{L(0, \alpha(w)) + p_{\alpha(w)}, \lambda^{u,v}(0, \beta(w)) + p_{\beta(w)}\}.$$

Further, we have

$$\lambda^{u,v}(l_k, n) = \begin{cases} p_{l_k} + L(\gamma(l_k), n) \\ \quad \text{if } v \text{ was the last operation on its machine,} \\ p_{l_k} + \max\{L(\gamma(l_k), n), L(\delta(v), n)\} \\ \quad \text{otherwise,} \end{cases}$$

for $w \in \{u, l_1, \ldots, l_{k-1}\}$,

$$\lambda^{u,v}(w, n) = p_w + \max\{L(\gamma(w), n), \lambda^{u,v}(\delta(w), n)\},$$

and

$$\lambda^{u,v}(v, n) = p_v + \max\{L(\gamma(v), n), \lambda^{u,v}(u, n)\}.$$

The evaluation $\lambda^{u,v}(0, n)$ of the impact of an interchange can be calculated in $O(|Q|)$ time, which is a substantial gain over the $O(n)$ time needed to calculate $\lambda^{u,v}(0, n)$. The quality of the approximation $\lambda^{u,v}(0, n)$ can be described by saying that $\lambda^{u,v}(0, n)$ differs from $L^{u,v}(0, n)$ only occasionally, and then not by much.

To introduce the next ingredient of our procedure, the guideposts, we need to establish some properties of the interchange procedure.

PROPOSITION 3.1. *If an interchange is performed on a critical pair u, v with the outcome that $L^{u,v}(0, n) > L(0, n)$, then after the interchange every critical path contains the arc (v, u).*

PROOF. Suppose a forward interchange is performed on u, v, with the postulated outcome. Since the makespan has increased, every critical path must contain at least one of the arcs of $S^{u,v} \backslash S$, i.e., $(l_1, u), \ldots, (l_k, u), (v, u)$. Now if (v, u) does not belong to some critical path, then (l_i, u) does for some $i \in \{1, \ldots, k\}$ since there is a path through l_i, v and u; but then $p_{l_i} > p_{l_i} + p_v$, a contradiction. A similar argument holds for the case of a backward interchange. □

PROPOSITION 3.2. *Suppose that a forward interchange is performed on a critical pair u, v, with the outcome that $L^{u,v}(0, n) > L(0, n)$. If $L(\gamma(v), n) \geq L(\gamma(u), n)$, then*

$$L^{u,v}(0, v) > L(0, v) - p_u; \tag{3.1}$$

and if $L(\gamma(v), n) < L(\gamma(u), n)$, then

$$L^{u,v}(v, n) > L(v, n) + p_u. \tag{3.2}$$

PROOF. Suppose $L(\gamma(v), n) \geq L(\gamma(u), n)$. Adding to both sides $L(0, v) + p_v$ yields

$$\begin{aligned} L(0, v) + p_v + L(\gamma(u), n) &\leq L(0, v) + p_v + L(\gamma(v), n) \\ &= L(0, n) < L^{u,v}(0, n), \end{aligned} \tag{3.3}$$

where the equality follows from the fact that $\gamma(v) \in P(0, n)$.

Now since from Proposition 3.1, (v, u) is on every critical path in $D_{S^{u,v}}$,

$$\begin{aligned} L^{u,v}(0, n) = L^{u,v}(0, v) + p_v + p_u \\ + \max\{L^{u,v}(\gamma(u), n), L^{u,v}(\delta(v), n)\}. \end{aligned}$$

Substituting this into (3.3) yields

$$\begin{aligned} L^{u,v}(0, v) &> L(0, v) - p_u + L(\gamma(u), n) \\ &\quad - \max\{L^{u,v}(\gamma(u), n), L^{u,v}(\delta(u), n)\} \\ &\geq L(0, v) - p_u, \end{aligned}$$

where the last inequality can be shown to hold by using two simple but different arguments, depending on where the maximum of the bracketed expression is attained.

Suppose now that $L(\gamma(v), n) < L(\gamma(u), n)$. Since u, v was a forward pair, $L(\gamma(v), n) \geq L(\delta(v), n)$ and hence $L(\gamma(u), n) > L(\delta(v), n)$, where $\delta(v)$, the machine successor of v before the interchange, is the machine successor of u after the interchange. Since from Proposition 2.4 the longest paths from $\gamma(u)$ to n and $\delta(v)$ to n are not affected by the interchange,

$$L^{u,v}(\gamma(u), n) > L^{u,v}(\delta(v), n),$$

and hence $\gamma(u)$ is on all longest paths from u to n. It then follows from Proposition 3.1 that after the interchange both arcs (v, u) and $(u, \gamma(u))$ belong to all critical paths from 0 to n, and hence

$$\begin{aligned} L^{u,v}(v, n) &= p_v + p_u + L^{u,v}(\gamma(u), n) \\ &> p_v + p_u + L^{u,v}(\gamma(v), n) \\ &= L(v, n) + p_u. \end{aligned}$$

Here the inequality follows from Proposition 2.4 and the assumption that $L(\gamma(u), n) > L(\gamma(v), n)$. □

An analogous result hold for backward interchanges, namely:

PROPOSITION 3.3. *Suppose that a backward interchange is performed on a critical pair u, v with the outcome that $L^{u,v}(0, n) > L(0, n)$. If $L(0, \alpha(v)) + p_{\alpha(v)} \leq L(0, \alpha(u)) + p_{\alpha(u)}$, then*

$$L^{u,v}(u, n) > L(u, n) - p_v; \tag{3.4}$$

and if $L(0, \alpha(v)) + p_{\alpha(v)} > L(0, \alpha(u)) + p_{\alpha(u)}$, then

$$L^{u,v}(0, u) > L(0, u) + p_v. \tag{3.5}$$

PROOF. Parallels that of Proposition 3.2. □

The interpretation of these propositions is pretty straightforward. Suppose a forward interchange results in an increase of the makespan. Then if (3.1) holds, at least part of the increase occurs on the path segment between 0 and v, and is attributable to the fact that although u was moved after v, v could not be "shifted to the left," i.e. started earlier, by the full amount p_u. On the other hand, if (3.2) holds, at least part of the increase occurs on the path segment between v and n, and is attributable to the fact that although u is the only operation inserted after v, nevertheless this insertion has added more than p_u to the length of the path from v to n. A similar interpretation applies to inequalities (3.4) and (3.5) in the case of a backward interchange.

It should be noted that inequalities (3.1) and (3.2) in the case of a forward interchange, as well as inequalities (3.4) and (3.5) in the case of a backward interchange, are not mutually exclusive; i.e., the increase in the length of a critical path after an interchange need not be restricted to one of the above two path segments.

Based on these propositions, after a forward interchange that increases the makespan we say that we have reached a *left guidepost* if inequality (3.1) holds, and a *right guidepost* if inequality (3.2) holds. Similarly, after a backward interchange that increases the makespan, we say that we have reached a *right guidepost* if inequality (3.4) holds, and a *left guidepost* if (3.5) holds.

As a result, after an interchange that increases the makespan, if we reach a left guidepost only, we restrict the list of candidates for an interchange to those critical pairs on a critical path segment between 0 and v; and if we reach a right guidepost only, we restrict the list of candidates to the critical pairs on a critical path segment between v and n (since (v, u) has just been fixed and is on the critical path, the segment actually considered in that between u and n). If both guideposts are reached, no restriction is applied to the list of candidates.

4. Neighborhood Trees and the Guided Local Search

A central problem of any search procedure based on local improvements is how to avoid being trapped at a local optimum. This is achieved by accepting nonimproving moves. The two well-known frameworks that have been used to do this are simulated annealing and tabu search. In this paper we propose a third framework for overcoming local optima, which we call *neighborhood trees*.

Our guided local search (GLS) procedure uses the neighborhood structure described in §2, and is guided by the evaluations and guideposts discussed in §3. But to overcome local optima, it uses neighborhood trees. In fact, the whole procedure is organized around the concept of neighborhood trees. Each node of a neighborhood tree corresponds to a complete selection S^i and associated schedule $t^i := t(S^i)$. Each edge of a neighborhood tree joins a parent node S^i to a child S^j, where S^j is obtained from S^i by performing an interchange on some critical pair.

The *operative neighborhood* $N(S^i)$ of a selection S^i is defined as the entire neighborhood if $t^i_n \leq t^{\pi(i)}_n$, where $S^{\pi(i)}$ is the parent of S^i (i.e. the selection from which S^i was obtained by an interchange), and as the neighborhood of S^i restricted by the guideposts if $t^i_n > t^{\pi(i)}_n$ (guideposts are available only if the interchange increases the makespan).

When at a given node S^i an interchange is performed on a pair u, v, the arc (v, u) gets fixed and remains so for all descendants of S^i. Arcs not fixed are termed free.

The children of a node S^i are the free members of $N(S^i)$, ranked by their evaluations and restricted to the first (in the order of the ranking) $f(l)$ nodes, where $f(l)$ is a decreasing function of the level l of the node S^i in the neighborhood tree. Let $\{u_1, v_1\}, \{u_2, v_2\}, \ldots, \{u_q, v_q\}$ with $q = f(l)$, be the ordered (ranked) list of critical pairs whose interchange produced the children of S^i. Then the first child is obtained by performing an interchange

on the pair u_1, v_1 and fixing the arc (v_1, u_1); the second child, by fixing the arc (u_1, v_1), then performing an interchange on the pair u_2, v_2, and fixing the arc (v_2, u_2); finally, the kth child for $k = 3, \ldots, q$ is obtained by fixing the arcs $(u_1, v_1), \ldots, (u_{k-1}, v_{k-1})$, then performing an interchange on the pair u_k, v_k and fixing the arc (v_k, u_k).

The fixing of arcs is, besides the function $f(l)$ that limits the number of children of a node, a second factor limiting the size of the tree. Finally, there is a third limiting factor in the form of a bound on the depth of the tree, i.e., on the number of levels allowed, which is a logarithmic function of the number of operations. Children are generated for every node in the neighborhood tree up to the specified depth.

As a result of these limiting factors, the size of a neighborhood tree is bounded by a linear function of the number of operations.

The GLS Procedure consists of a sequence of variable depth forays into the set of feasible schedules, where each foray generates a neighborhood tree. We call such a foray a cycle. (There is a one-to-one correspondence between cycles and neighborhood trees.) Let $T_1, \ldots, T_k$ be the sequence of neighborhood trees to be generated. The root node of T_1 is a feasible schedule that can be constructed by any approximation procedure. The root node of any other neighborhood tree T_j, $j \neq 1$, is obtained as follows. As a neighborhood tree is being generated, the k best (in terms of makespan) among those of its nodes separated from the root by at least $\lceil \log_2 mn^* \rceil$ interchanges are stored (here m and n^* are the number of machines and jobs, respectively). If the level $\lceil \log_2 mn^* \rceil$ is not reached, the best node is stored. Now if the neighborhood tree T_{j-1} has produced an improvement in the current best makespan, then the root of T_j is that node of T_{j-1} associated with the best makespan. Otherwise, one of the nodes stored for T_{j-1} is selected through a biased random choice (i.e. one that assigns a probability of $1/2^i$ to the ith best node), and made the root of T_j.

The standard version of GLS, called GLS/1, starts with a schedule generated by a randomized MWKR (most work remaining) dispatching rule, i.e., one that orders the jobs according to MWKR and schedules the ith one with a probability of $1/2^i$. It then runs for as long as it keeps obtaining improvements, and stops after k cycles without an improvement. Here k is a parameter set by the user (in our computational tests reported in §6, we used $k = 120$).

Versions GLS/2, GLS/3, and GLS/4 are the same as GLS/1 applied two, three, and four times.

5. Embedding GLS into the Shifting Bottleneck Procedure

It is in the nature of local search procedures that they often get trapped in a region around some local optimum, and their ability to break out of such entrapments and find improved local optima depends very much on their neighborhood structure. This suggests that combining procedures with different neighborhood structures may help overcome entrapments and carry the search to areas of the feasible set which none of the procedures, run in isolation, would visit. We therefore decided to attempt such an approach, and embedded the GLS whose neighborhood definition is based on interchanges on critical pairs of operations anywhere on a critical path or path segment, into the Shifting Bottleneck (SB) Procedure, whose neighborhood concept is based on the optimization of sequences on individual machines.

This is done as follows. The SB Procedure sequences the machines one at a time, consecutively. To do this, for each machine not yet sequenced it solves to optimality a one-machine scheduling problem that is a relaxation of the original problem, and uses the outcome both to rank the machines, and to sequence the machine with highest rank (largest makespan), termed the bottleneck. Every time a new machine has been sequenced, the procedure reoptimizes the sequence of each previously processed machine by again solving a one-machine problem, with the sequence on the other machines held fixed. If the cycle of reoptimizations is pursued until no more improvement is obtained, the resulting schedule is a local optimum over a neighborhood defined as the set of all schedules obtainable from a given schedule by changing the sequence (in any way) on any single machine.

Notice that, as a byproduct of the local optimization, the SB procedure also generates a lower bound on the makespan. Namely, the makespan of each one-machine problem solved before fixing the sequence on any machine is obviously a lower bound on the makespan of

the original job shop scheduling problem, and the makespan of the one-machine problem for the bottleneck machine is the strongest among these lower bounds.

We embed the GLS into the SB Procedure by replacing the reoptimization cycle of the SB with a number of cycles of the GLS Procedure. GLS is applied after removing the disjunctive arcs corresponding to machines not yet sequenced. Each time we generate 2θ cycles (neighborhood trees), where $\theta = \lceil \log_2(m_0 n^*) \rceil$, with $m_0 = |M_0|$ denoting the number of fixed machine sequences and n^* the number of jobs. The root node of the first tree is the partial schedule defined by the selections on those machines already sequenced (indexed by M_0). The root nodes of subsequent trees are chosen as described in the previous section. The best schedule obtained from the 2θ neighborhood trees is then used as a starting point for the continuation of the SB Procedure. We call the combined Procedure SB-GLS1 (Shifting Bottleneck with Guided Local Search, version 1).

In another version called SB-GLS2, we amend the above procedure with some further local search. Namely, after all the machines have been sequenced, i.e., $M_0 = M$, we execute the following procedure, called GLS*. We remove from each machine in turn its selection, and apply the GLS Procedure for θ cycles to the resulting $(m-1)$-machine problem; then we apply for 2θ cycles the GLS Procedure to the full m-machine problem obtained by introducing (and sequencing) the removed machine into the current best selection (for the $(m-1)$ machine problem). SB-GLS2 is repeated until there is no improvement.

Finally, in a third version the procedure is further amended as follows. In order to move to a point in the feasible set far enough from the current locus of the search and yet of good quality, we take the current best schedule and select $\lfloor m^{1/2} \rfloor$ machines for deletion (sequence removal) by a random choice biased in favor of machines sequenced early in the SB-GLS1 Procedure (namely, the ith machine sequenced is chosen with a probability of $(2/3)^i$). The remaining partial schedule (with $m - \lfloor m^{1/2} \rfloor$ machines) is then taken as the root node of a neighborhood tree, and GLS is applied for θ cycles. Next, the schedule is completed by the SB-GLS1 Procedure and when all the machines have been scheduled, i.e., $M_0 = M$, we execute the procedure GLS* described above. This procedure (of removing the sequence of $\lfloor m^{1/2} \rfloor$ randomly selected machines and then executing the above described steps), which can be repeated several times, is called RGLSk, where R stands for random and k is the number of times the procedure is applied. We call this version of the entire procedure SB-RGLSk.

As mentioned above, the first round of one-machine problems solved by the SB procedure generates a lower bound on the makespan of the original problem. If at any time during the rest of the procedure, in any of its versions, a schedule is generated whose makespan matches the lower bound, that makespan is of proven optimal value. One would be inclined to think that this does not happen too often, since the lower bound is derived from a one-machine relaxation of the problem. Surprisingly, however, this lower bound is the value of the actual optimum in 57% of the test problems run in our experiments (see §6).

6. Computational Results

The procedures of §§4 and 5 were implemented in C and tested on a large number of job shop scheduling problems from the literature. The results obtained with our procedures were then compared with results obtained with other procedures.

First we discuss the job shop scheduling instances we use in our computational comparisons. Problems LA1-40 are the 40 instances generated by Lawrence (1984). Problems ORB1-10 were generated by Applegate and Cook (1991). Problems TD1-50 were generated by Taillard (1993). Finally, problems ABZ5-9 were generated by Adams et al. (1988).

We compare our procedures with all procedures for which we could find results (makespan, CPU time) in the literature. We use the following notation for those procedures: LAL stands for a simulated annealing procedure of Laarhoven et al. (1992). BC stands for a taboo search procedure of Barnes and Chambers (1995). DT stands for a taboo search procedure of Dell'Amico and Trubian (1993) and gives the outcome (best solution) of five runs. NS(1) and NS(3) stand for the taboo search procedure of Nowicki and Smutnicki (1993); NS(1) gives the outcome of a single run, and NS(3) gives the outcome of three runs. DP stands for a genetic algorithm

of Dorndorf and Pesch (1995). AC stands for the approximation procedure (Shuffle) of Applegate and Cook (1991), which first generates a modified Shifting Bottleneck schedule, then fixes some of the machines and optimizes over the rest. TD stands for the taboo search procedure of Taillard (1993). The results for these procedures are from the corresponding publications listed above, except for Applegate and Cook (1991), for which the results are from runs performed by Vaessens et al. (1994).

Whenever available, we give the best known lower bound (denoted LB) for the problem. For the problems LA1-40 and ORB1-10, these bounds are taken from Applegate and Cook (1991). Whenever the LB is available, we calculate the relative error for each procedure and each instance; i.e., the percentage by which the solution obtained is above the best known LB: $100(UB/LB)$. MRE denotes the mean relative error for the set of problems reported.

The acronym CI-CPU stands for computer-independent CPU times. These are based on comparisons of Dongarra (1993), as interpreted by Vaessens et al. (1994).

First, we tested our procedures on the 40 problem instances LA1-40. These problems range between 5 to 15 machines and 10 to 30 jobs. The results are shown in Tables 1–3. A blank entry in any table means no improvement over the corresponding entry of the previous column.

The problems LA6-10 (15 jobs, 5 machines), LA11-15 (20 jobs, 5 machines), and LA30-35 (30 jobs, 10 machines) are easy because the number of jobs is several times larger than the number of machines. They were solved to optimality by both GLS and SB-GLS in a few seconds (with less time required by SB-GLS), and the results are ommitted from our tables.

Table 1 compares the performance of our procedures with each other and with DP, LAL, and DT, first on the still relatively easy 10 jobs, 5 machines (LA01-05) problems, then on the more difficult 10 machines (LA16-30) and 15 machines (LA36-40) problems. On these problems, GLS/1 clearly outperforms DP and LAL, both in terms of solution quality and computing time. It is on the average slightly weaker in solution quality than DT, but requires only half as much computing time. The major message of Table 1, however, is that SB-GLS significantly outperforms GLS/1 and all the other procedures both in terms of average solution quality and computing time. We note that, although the table does not show this, in a number of cases SB-GLS was able to prove optimality by using its self-generated lower bounds.

To make a more detailed comparison between the procedures on problems for which the comparison is meaningful, we selected the 12 most difficult instances among the 40 LA problems. Among these, the instances LA21, 27, 29, and 38 are viewed by the literature (see for instance Vaessens et al. 1994) as challenging, since their optimal solutions were unknown until now although every known algorithm has been tried on them. To these 12 instances we have added FT10, the notorious 10 jobs, 10 machines problem of Fisher and Thompson (1963). Table 2 contains a detailed comparison of the performance of our two procedures, GLS and

Table 1 Mean Relative Error (MRE) over Best Known Lower Bound (=100) (CI-CPU times in parenthesis)

Problem Class	Jobs/Machines	DP	LAL	DT	GLS/1	SB-GLS1	SB-GLS2
LA01-05	10/5	100.57	100.30	100.00	100.00	100.44	100.00
		(22.14)	(115.52)	(36.40)	(14.69)	(2.99)	(14.78)
LA16-20	10/10	101.40	100.71	100.00	100.09	100.50	100.00
		(124.16)	(679.44)	(178.70)	(54.06)	(24.25)	(117.58)
LA21-25	15/10	102.00	101.35	100.45	100.67	100.74	100.26
		(218.24)	(1,990.86)	(382.75)	(127.18)	(53.31)	(349.10)
LA26-30	20/10	102.51	102.43	101.22	100.80	101.24	100.87
		(372.19)	(4,103.05)	(377.10)	(226.19)	(51.56)	(304.06)
LA36-40	15/15	104.80	101.87	100.98	101.52	100.99	100.43
		(543.04)	(5,177.88)	(605.80)	(252.55)	(126.04)	(694.95)

Table 2 **Makespan and CI-CPU Time (in parenthesis) for 13 Hard Problems**

	FT10	LA02	LA19	LA21	LA24	LA25	LA27	LA29	LA36	LA37	LA38	LA39	LA40	MRE
LB	930	655	842	1,040	935	977	1,235	1,120	1,268	1,397	1,184	1,233	1,222	100.00
CODES	938	655	842	1,046	965	992	1,269	1,191	1,275	1,422	1,267	1,257	1,238	102.14
AC	(61.8)	(19.8)	(182)	(>10^6)	(>10^6)	(246)	(1,511)	(>10^5)	(8,370)	(3,943)	(>10^5)	(>10^5)	(375)	
DP	938	666	863	1,074	960	1,008	1,272	1,204	1,317	1,484	1,251	1,282	1,274	103.74
	(171)	(26.2)	(124)	(216)	(220)	(215)	(388)	(386)	(537)	(561)	(537)	(524)	(557)	
LAL	951	655	848	1,063	952	992	1,269	1,218	1,293	1,433	1,215	1,248	1,234	102.26
	(740)	(111)	(788)	(1,891)	(1,993)	(2,026)	(4,308)	(4,188)	(5,079)	(5,023)	(5,206)	(5,478)	(5,104)	
BC	935	655	843	1,053	946	988	1,256	1,194	1,278	1,418	1,211	1,237	1,239	101.45
	(190)	(347)	(2,605)	(2,082)	(332)	(2,113)	(2,978)	(2,347)	(2,657)	(2,784)	(2,171)	(3,104)	(1,074)	
DT	935	655	842	1,048	941	979	1,242	1,182	1,278	1,409	1,203	1,242	1,233	101.01
	(390)	(47)	(260)	(497)	(455)	(479)	(636)	(704)	(596)	(606)	(642)	(595)	(592)	
NS(1)	930	655	842	1,055	948	988	1,259	1,164	1,275	1,422	1,209	1,235	1,234	101.19
	(15)	(4)	(30)	(10.5)	(92)	(77.5)	(33)	(247)	(312)	(222)	(82.5)	(163)	(161)	
NS(3)	930	655	842	1,047	939	977	1,236	1,160	1,268	1,407	1,196	1,233	1,229	100.54
GLS/1	930	655	846	1,053	950	982	1,244	1,157	1,269	1,425	1,213	1,233	1,260	101.19
	(85.5)	(18.3)	(51.8)	(171)	(119)	(179)	(312)	(389)	(182)	(235)	(271)	(385)	(191)	
SB-GLS1	930	666	852	1,048	941	993	1,243	1,182	1,268	1,397	1,208	1,249	1,242	101.31
	(32.3)	(3.25)	(29.8)	(63)	(64.3)	(65.5)	(75)	(110)	(139)	(93.3)	(140)	(121)	(141)	
SB-GLS2	930	655	842	1,048	937	977	1,240	1,164	1,268	1,397	1,198	1,233	1,234	100.57
	(113)	(18.8)	(184)	(281)	(437)	(560)	(524)	(923)	(447)	(366)	(748)	(1,084)	(830)	
SB-RGLS1	930	655	842	1,048	937	977	1,235	1,164	1,268	1,397	1,198	1,233	1,226	100.49
	(214)	(18.8)	(284)	(541)	(637)	(755)	(788)	(1,258)	(781)	(792)	(1,180)	(1,376)	(1,959)	
SB-RGLS2	930	655	842	1,046	935	977	1,235	1,164	1,268	1,397	1,196	1,233	1,224	100.43
	(618)	(18.8)	(673)	(1,529)	(1,704)	(1,540)	(788)	(2,655)	(2,300)	(2,055)	(3,203)	(2,828)	(3,974)	

SB-GLS (four versions of the latter), against that of the procedures AC, DP, LAL, BC, DT, and NS (two versions of the latter). The computing times are of course total times, except for the code BC for which the computing times are those needed to obtain the best solution (total times for BC are not reported). Computing times for NS(3) are not reported in Nowicki and Smutnicki (1993).

Overall, procedure SB-GLS1 outperforms procedures AC, DP, LAL, and BC both in terms of solution quality and computing time. SB-GLS2 outperforms DT in solution quality while matching it in computing time, and obtains solutions of roughly the same quality as NS(3) and better than NS(1), in somewhat longer computing time. Finally, SB-RGLS1 and SB-RGLS5 further improve the schedules and obtain a mean relative error of 0.43%. This result, however, is obtained at considerably higher computational cost.

It is to be noted that SB-GLS1 found the optimal solution of FT10 (without, however, proving optimality) in just 13 seconds on the SUNSPARC-330. Among the nine instances whose optimal solution value was known, our procedures found optimal solutions for all but one (the exception being LA40). Further, SB-RGLS1 also found the optimal solution of LA27 (unknown before), and was able to prove optimality based on the criterion discussed in §5. Finally, we notice that GLS/1 found a solution with value 1157 for problem LA29, better than the best previously known value.

Table 3 lists the best known lower and upper bounds on the makespan of the four most difficult problems of the LA set and FT10, along with the procedures that found the upper bounds and the current status of the problem.

Table 4 shows our computational results on the problems TD1-50 generated by Taillard. They consist of ten instances in each of five classes, where the jobs range from 15 to 30, and the machines from 15 to 20. For these problems, Taillard (1993) reports the makespan obtained after several runs of his taboo search procedure. Nowicki and Smutnicki (1993) also ran these problems, and report for each of the five problem classes the mean relative performance of their

procedure (in terms of makespan) as compared to TD. We report for each class of problems, the mean relative performance in terms of makespan (MRP), where the relative performance is 100 times the ratio between the makespan obtained by the given procedure and TD. Taillard has generated three additional classes of problems, with 50, 50, and 100 jobs and 15, 20, and 20 machines, respectively, but we omit them as too easy (as explained earlier, problems with many more jobs than machines are easy).

Our procedures improved upon the average makespan found by both TD and NS in each of the five problem classes. A comparison of the computational effort for these runs is not possible, since neither Taillard (1993) nor Nowicki and Smutnicki (1993) report computing times.

Table 3 Difficult Problems

Problem	Best Known Lower Bound on Makespan	Best Known Makespan	Found by Procedure	Status of Problem
FT10	930	930	NS(1), GLS/1, SB-GLS1	SOLVED
LA21	1,040	1,046	AC, SB-RGLS3	OPEN
LA27	1,235	1,235	SB-RGLS1	SOLVED
LA29	1,120	1,157	GLS/1	OPEN
LA38	1,184	1,196	NS(3), SB-RGLS4	OPEN

Table 4 Mean Relative Performance (MRP) on the Problems of TD1-50 (TD = 100)

Problem Class	Jobs/Machines	TD	NS	SB-GLS1	SB-GLS2	SB-RGLS1	SB-RGLS5	SB-RGSL10
TD1-10	15/15	100	100.8	100.63	100.08	99.94	99.73	99.66
TD11-20	20/15	100	100.9	101.59	100.30	99.91	99.67	99.50
TD21-30	20/20	100	101.2	101.36	100.37	100.24	100.03	99.90
TD31-40	30/15	100	100.6	101.07	99.19	99.08	99.08	98.89
TD41-50	30/20	100	101.9	102.05	100.20	99.89	99.57	99.44

Table 5 Our Procedures Versus Optimizing (Applegate and Cook, 1991)

	Branch and Bound [AC91]			SB-GLS1		SB-GSL2		SB-RGLS5	
Problem	Makespan (Optimal)	Nodes	CI-CPU	Makespan	CI-CPU	Makespan	CI-CPU	Makespan	CI-CPU
ORB1	1,059	71,812	2,076	1,059	43.2				
ORB2	888	153,578	3,478	892	33.0	888	221		
ORB3	1,005	130,181	3,217	1,005	40.6				
ORB4	1,005	44,547	1,419	1,024	30.5	1,019	185	1,013	714
ORB5	887	23,113	736	889	38.1				
ORB6	1,010	–	–	1,019	30.5	1,010	312		
ORB7	397	–	–	401	30.5	307	173		
ORB8	899	–	–	919	25.4	899	244		
ORB9	934	–	–	953	27.9	934	183		
ORB10	944	–	–	944	35.6				

– not reported.

Problems ORB1-10 have been solved to optimality by Applegate and Cook (1991). These are instances of 10 jobs and 10 machines. In Table 5, we report the results for our procedures along with those of the optimization algorithm of Applegate and Cook (1991). Notice that a mean relative error of 0.10% has been obtained in a fraction of the time required by the exact method, and for eight of the ten problems, the optimal solution was found.

Additional details on the computational experience reported here, as well as computational results on additional problems generated by Storer et al. (1992) and by Yamada and Nakano (1992) are to be found in Balas and Vazacopoulos (1994).

In conclusion, the hybrid procedure which embeds GLS into a Shifting Bottleneck framework turns out to be particularly efficient. It finds an optimal solution for the notorious ten machines ten jobs scheduling problem of Fisher and Thompson (1963) in 12.94 seconds. It finds an optimal solution to all but one of the problems in the set LA1-40 for which the optimum is known. Additionally, it finds an optimal solution to eight out of ten problems ORB1-10. It finds improved solutions to most of the TD1-50 problems, and in many cases it finds an optimal solution and proves optimality. Finally, it improves the current best solutions for the most difficult problems of the set LA1-40. We conclude that the heuristic SB-GLS shows robustness in all categories and provides schedules of high quality in a reasonable amount of computing time.[1]

[1] Thanks are due to Rob Vaessens and three anonymous referees for comments that helped improve the paper. The first author's research was supported by the National Science Foundation through grants DMI-9424348 and DMS-9509581, and by the Office of Naval Research through contract N00014-89-J-1063.

References

Aarts, E. H. L., P. J. M. van Laarhoven, J. K. Lenstra, and N. L. J. Ulder, "A Computational Study of Local Search Algorithms for Job Shop Scheduling," *ORSA J. Computing*, 6 (1994), 108–117.

Adams, J., E. Balas, and D. Zawack, "The Shifting Bottleneck Procedure for Job Shop Scheduling," *Management Sci.*, 34, 3 (1988), 391–401.

Applegate, D. and W. Cook, "A Computational Study of Job-Shop Scheduling," *ORSA J. Computing*, 3, 2 (1991).

Baker, K., *Introduction to Sequencing and Scheduling*, Wiley, New York, 1974.

Balas, E., "Machine Sequencing via Disjunctive Graphs: An Implicit Enumeration Algorithm," *Oper. Res.*, 17 (1969), 941–957.

——, J. K. Lenstra, and A. Vazacopoulos, "The One Machine Scheduling Problem with Delayed Precedence Constraints and its Use in Job Shop Scheduling," *Management Sci.*, 41 (1995), 94–109.

—— and A. Vazacopoulos, "Guided Local Search with Shifting Bottleneck for Job Shop Scheduling," MSRR 609, GSIA, Carnegie Mellon University, Pittsburgh, PA, October, 1994 (revised November 1995).

Barnes, J. W. and J. B. Chambers, "Solving the Job Shop Scheduling Problem Using Tabu Search," *IIIE Trans.*, 27 (1995), 257–263.

Blazewicz, J., K. Ecker, G. Schmidt, and J. Węglarz, *Scheduling in Computer and Manufacturing Systems*, Springer-Verlag, 1992.

Carlier, J., "The One-machine Sequencing Problem," *European J. Oper. Res.*, 11 (1982), 42–47.

—— and E. Pinson, "An Algorithm for Solving the Job Shop Problem," *Management Sci.*, 35 (1989), 164–176.

Conway, R. N., W. L. Maxwell, and L. W. Miller, *Theory of Scheduling*, Addison-Wesley, Reading, MA, 1967.

Dongarra, J. J., "Performance of Various Computers using Standard Linear Equations Software," Report CS-89-85, Computer Science Department, University of Tennessee, Knoxville, TN, 1993.

Dell'Amico, M. and M. Trubian, "Applying Tabu-Search to the Job-Shop Scheduling Problem," *Ann. Oper. Res.*, 41, 1–4 (1993), 231–252.

Dorndorf, U. and E. Pesch, "Evolution Based Learning in a Job Shop Scheduling Environment," *Comput. Oper. Res.*, 22 (1995), 25–40.

Fisher, H. and G. L. Thompson, "Probabilistic Learning Combinations of Local Job Shop Scheduling Rules," in J. F. Muth, G. L. Thompson (eds.), *Industrial Scheduling*, Prentice-Hall, Englewood Cliffs, NJ, 1963.

French, S., *Sequencing and Scheduling: An Introduction to the Mathematics of the Job Shop*, Wiley, New York, 1982.

Grabowski, J., E. Nowicki, and S. Zdrzalka, "A Block Approach for Single-machine Scheduling with Release Dates and Due Dates," *European J. Oper. Res.*, 26 (1986), 278–285.

van Laarhoven, P. J. M., E. H. L. Aarts, and J. K. Lenstra, "Job Shop Scheduling by Simulated Annealing," *Oper. Res.*, 40 (1992), 113–125.

Lageweg, B. J., J. K. Lenstra, and A. H. G. Rinnooy Kan, "Job-Shop Scheduling by Implicit Enumeration," *Management Sci.*, 24 (1977), 441–450.

Lawrence, S., Supplement to "Resource Constrained Project Scheduling: An Experimental Investigation of Heuristic Scheduling Techniques," GSIA, Carnegie Mellon University, Pittsburgh, PA, 1984.

Lenstra, J. K., "Sequencing by Enumerative Methods," Mathematical Centre Tract 69, Mathematisch Centrum, Amsterdam, The Netherlands, 1976.

Matsuo, H., C. J. Suh, and R. S. Sullivan, "A Controlled Search Simulated Annealing Method for the General Jobshop Scheduling

Problem," Working Paper 03-04-88, 1988, Department of Management, Graduate School of Business, The University of Texas at Austin, Austin, TX, 1988.

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

Nowicki, E. and C. Smutnicki, "A Fast Taboo Search Algorithm for the Job Shop Scheduling Problem," Instytut Cybernetyki Technpicznej Politechniki Wrclawskiej Raport serii: Preprinty nr August 1993.

Potts, C. N., "Analysis of a Heuristic for One Machine Sequencing Problems with Release Dates and Delivery Times," *Oper. Res.*, 28, 6 (1980), 1436–1441.

Rinnooy Kan, A. H. G., "Machine Scheduling Problems: Classification, Complexity and Computations," Nijhoff, The Hague, Netherlands, 1976.

Storer, R. H., S. David Wu, and Renzo Vaccari, "New Search Spaces for Sequencing Problems with Application to Job Shop Scheduling," *Management Sci.*, 39, 10 (1992), 1495–1509.

Taillard, E., "Benchmarks for Basic Scheduling Problems," *European J. Operational Res.*, 64 (1993), 278–285.

——, "Parallel Taboo Search Techniques for the Job Shop Scheduling Problem," *ORSA J. Computing*, 6 (1994), 118–125.

Vaessens, R. J. M., E. H. L. Aarts, and J. K. Lenstra, "Job Shop Scheduling by Local Search," Memorandum COSOR 94-05, Eindhoven University of Technology, Department of Mathematics and Computing Science, Eindhoven, The Netherlands, 1994.

Yamada, T. and R. Nakano, "A Genetic Algorithm Applicable to Large-scale Job-shop Instances," in R. Manner, B. Manderick (eds.), *Parallel Instance Solving from Nature* 2, North-Holland, Amsterdam, The Netherlands, 1992, 281–290.

Accepted by Thomas M. Liebling; received December 1994. *This paper has been with the authors* 2 *months for* 2 *revisions.*