

# New Tabu Search Results for the Job Shop Scheduling Problem

John B. Chambers<sup>1</sup> and J. Wesley Barnes<sup>2</sup>

<sup>1</sup>*Current address: Department of Computer Sciences, TAY 2.124, The University of Texas at Austin, Austin, Texas 78712. Email: jbc@cs.utexas.edu*

<sup>2</sup>*Cullen Trust for Higher Education Endowed Professor in Engineering, Graduate Program in Operations Research and Industrial Engineering, Department of Mechanical Engineering, ETC 5.128D, The University of Texas at Austin, Austin, Texas 78712. Email: wbarnes@mail.utexas.edu*

---

## Abstract

In the classical job shop scheduling problem (JSSP),  $n$  jobs are processed to completion on  $m$  unrelated machines. Each job requires processing on each machine exactly once. For each job, technology constraints specify a complete, distinct routing which is fixed and known in advance. Processing times are sequence-independent, fixed, and known in advance. Each machine is continuously available from time zero, and operations are processed without preemption. The objective is to minimize the maximum completion time (makespan).

This report describes a dynamic, adaptive tabu search (TS) strategy for JSSP incorporating elements of an earlier investigation by the authors. Improved computational results for a set of reference problems are presented.

---

# New Tabu Search Results for the Job Shop Scheduling Problem

## 1. Introduction

In the classical job shop scheduling problem (JSSP),  $n$  jobs are processed to completion on  $m$  unrelated machines. Each job requires processing on each machine exactly once. For each job, technology constraints specify a complete, distinct routing which is fixed and known in advance. Processing times are sequence-independent, fixed, and known in advance. Each machine is continuously available from time zero, and operations are processed without preemption. The objective is to minimize the maximum completion time (makespan).

The general JSSP is strongly NP-hard [Garey, Johnson, and Sethi 1976; Garey and Johnson 1979], and has been the subject of numerous heuristic techniques (see, e.g., French [1982]). Methods based on local search have shown special promise [Vaessens, Aarts, and Lenstra 1994].

Tabu search (TS) is a local search metaheuristic which relies on specialized memory structures to avoid entrapment in local minima and achieve an effective balance of intensification and diversification. TS has proved remarkably powerful in finding high-quality solutions to computationally difficult combinatorial optimization problems drawn from a wide variety of applications [Glover and Laguna 1993]. We assume that the reader is familiar with the basic principles of TS [Glover 1989, 1990, 1994; Glover and Laguna 1993; Glover, Taillard, and deWerra 1993].

TS results in the area of production scheduling have been especially successful [Barnes and Laguna 1991]. In particular, we have previously reported experience with a simple but highly effective TS approach to JSSP [Barnes and Chambers 1991a, 1991b, 1992a, 1992b, 1995]. In this paper, we describe a dynamic, adaptive TS strategy yielding results significantly superior to those of our earlier investigations, and competitive with or superior to a well-known randomized dynamic TS implementation [Dell'Amico and Trubian 1993].

## 2. Selected Literature

Barnes and Chambers [1991a, 1991b] describe a prototype TS implementation for JSSP. An initial solution is obtained by selecting the best solution (i.e., the one with minimum makespan) from among fourteen priority dispatching schedules. The *disjunctive graph* representation [Roy and Sussman 1964] of the scheduling problem gives rise to two important observations [Balas 1969; van Laarhoven, Aarts, and Lenstra 1992], namely, that: 1) given a feasible solution, the exchange of two adjacent critical operations (on the same machine) cannot yield an infeasible solution; and 2) the makespan can be improved only by performing such exchanges. These insights are used to define a search neighborhood. (This neighborhood was also explored in an earlier study by Taillard [1989, 1992, 1994].) At each move, the effect of each possible exchange is estimated using Balas'  $\Delta$  [Balas 1969]. The reversal of an operation pair exchanged during the extent of short term memory is forbidden unless the resulting makespan would be better than any

seen to date (a simple *aspiration criterion*). In the event that no admissible moves are available, all tabu restrictions are released and the search continues from the current solution.

The authors extend this strategy [Barnes and Chambers 1992a, 1992b, 1995] by using a solution stack (LIFO list) to diversify the search; this stack is a simple form of *historical generator* [Glover 1991]. (A variation of this list approach has been described more recently by Nowicki and Smutnicki [1993].) Each new best-to-date solution is “pushed” onto the stack when it is discovered. Subsequently, such solutions may be “popped” from the stack in turn as new incumbent solutions, from which an intensified search is performed in a *semidynamic* fashion over a prespecified range of short term memory values, within limits set on CPU time and on the maximum number of consecutive moves allowed with no improvement in makespan. Computational results for a set of standard test problems are reported, and compared with the “shifting bottleneck” heuristic of Adams, Balas, and Zawack [1988] and with a computational study by Applegate and Cook [1991].

A growing body of TS research suggests that superior results may be achieved by allowing the length of short term memory to vary *dynamically* in response to changing conditions of the search. Dell’Amico and Trubian [1993] describe their experience with a randomized, dynamic, adaptive TS implementation for JSSP. Short term memory is allowed to vary, within lower and upper limits, decreasing in response to improving moves and increasing in response to nonimproving moves. Every  $\Lambda$  iterations (where  $\Lambda$  is a prespecified constant), these lower and upper limits are randomly adjusted within prespecified ranges. A move is randomly selected if no admissible move is available. Repeated makespan values, coupled with so-called *witness* moves, help identify potential cycling behavior. Dell’Amico and Trubian report results consistently superior to the method of Adams, Balas, and Zawack [1988].

In the next section, we discuss a deterministic, dynamic, adaptive TS strategy for JSSP which exhibits significantly improved performance over our earlier method. Computational results are reviewed for a well-known set of test problems.

### 3. An Improved Tabu Search Procedure

#### *Initial solution*

A feasible initial solution is obtained by selecting from among twelve priority dispatching solutions. Six *active* and six *nondelay* schedules are computed using the priority rules SPT, LPT, LWKR, MWKR, MOPNR, and RATIO. The schedule with the smallest makespan is installed as the starting solution (*cf.* Barnes and Chambers [1995]).

To help identify “easy” problems for which the dispatching solution is optimal or nearly so, a trivial *a priori* lower bound is also computed. This bound is taken as the maximum over the total processing time required by each job and each machine. This bounding strategy is likely to be poor in many cases. In the modular spirit of tabu search, it is self-contained to facilitate easy replacement in experiments requiring a strong optimality test.

#### *Move evaluation*

The computation of makespan and the determination of critical operations is

performed using a straightforward CPM labeling procedure [see, e.g., Baker 1974]. The special structure of the job shop scheduling problem makes such a computation especially efficient [Adams, Balas, and Zawack 1988].

However, to enhance the performance of our previous implementation, the forward/backward labeling technique is reimplemented as a three-step procedure, using the disjunctive graph representation. The first step performs a topological sort of the graph, with no additional computation. The sorted graph is used in the second and third steps for the evaluation of early and late start and finish times. It is well known, given a directed acyclic graph  $G$  with valid topological sort  $T$ , that the reversal of  $T$  is a valid topological sort of the graph  $G'$  resulting from the reversal of each edge in  $G$  [see, e.g., Sedgewick 1983]. Thus, the CPM computations needed in step two (three) can be performed by a simple forward (backward) additive scan of the sort vector obtained at the end of step one. The forward scan computes makespan, and the backward scan computes the quantities needed for the determination of Balas'  $\Delta$  [Balas 1969]. A save/restore buffer is included to retain the previous set of operation times (CPM values) during the incremental evaluation of moves in a given iteration. Finally, our earlier experience confirms that it is important to allow zero-valued moves which traverse "flat" regions of the solution space.

The move evaluation procedure, as a component of the overall move selection process, is sketched in Figure 1.

### *Neighborhood*

A sequencing move is defined by the exchange of certain adjacent critical operation pairs. In our earlier investigation, we considered the exchange of every adjacent critical operation pair on every machine. However, if  $(cs', i, j, cs'')$  is a critical sequence on a given machine, then the interchange  $(i, j) \rightarrow (j, i)$  cannot improve the makespan [Suh 1988; Matsuo, Suh, and Sullivan 1988]. We therefore adopt a *reduced* neighborhood which considers the exchange of a pair of adjacent critical operations only if they occur in isolation, or at the head or tail of a critical sequence. As before, each machine is scanned successively for candidate exchange pairs.

### *Tabu attribute*

During the extent of short term memory, we forbid the reversal of the exchange of a critical operation pair by recording the iteration number on which the exchange was performed and requiring that this number plus the *current* length of short term memory be strictly less than the current iteration number. A record is also made, each time this tabu attribute is tested, as to whether a particular tabu move is the *oldest* tabu move; this information is used during the move selection procedure (see below).

A simple aspiration criterion, *viz.*, that the contemplated move would yield a makespan better than the best seen to date, can override a move's tabu status.

### *Memory structures*

In our earlier approach, we used a *semidynamic* short term memory strategy. For a given problem, the length of short term memory was iterated over a range of values; however, only one value at a time was used during each search phase.

As noted in Section 2, it may be possible to obtain superior results when short

term memory length is allowed to vary dynamically during the course of the search. For example, Taillard [1989, 1994] precomputes a range for short term memory length, based on problem size, and then periodically selects a new length randomly from this range. Another example is the somewhat more sophisticated approach of Dell’Amico and Trubian [1993], described previously.

In our current investigation, we allow the length of short term memory to adapt *dynamically* and *deterministically* in response to changing search conditions. A prespecified minimum, *stm\_lo*, and maximum, *stm\_hi*, bound the allowable range of short term memory. A strictly improving move, relative to the current search, causes the working short term memory length (if greater than *stm\_lo*) to be reduced by 1, in an attempt to focus the search in a region of potential improvement. However, a nonimproving move causes the working short term memory length (if less than *stm\_hi*) to be increased by 1, in an attempt to drive the search away from an apparently nonimproving region (see Figure 2).

Drawing on our earlier study, we utilize a solution stack (LIFO solution list) as a simple form of historical generator. As an implementation note, the “stack” is now maintained as a circular linked list which is “pushed” (“popped”) by allowing a list pointer to advance (retreat); the advancing list pointer overwrites old entries after STK\_N entries have been stored. For the test problems, a STK\_N of 30 was found to offer an appropriate balance between memory use and diversification while avoiding early search termination.

When an improving move is performed, the short term memory is adjusted as described above. If the makespan is better than the best seen to date, the solution is pushed onto the solution stack (see Figure 3). However, if a nonimproving move has been executed, and if a prespecified limit on the number of consecutive nonimproving moves has been exceeded, the solution on top of the solution stack is popped and installed as the new solution. Short term memory bounds are reset to their original values, tabu information and the solution history list (see below) are cleared, and the search is then resumed using the largest value of short term memory (see Figure 4).

### *Cycling strategy*

A structure similar to that used for the solution stack is defined to address regionalized cycling behavior. In practice, a simple test for repeated makespans tends to report false cycles, especially in problems characterized by many “flat” regions. Thus, it may be important to compare actual schedules if a sequence of repeated makespans has been detected. However, storage of a large number of explicit solutions may be impractical, depending on memory requirements, and is not necessarily desirable, since historical information about cycling in a localized region may be of limited interest after diversification strategies have forced the search to depart that region. Consequently, we implement a second circular linked list large enough to remember the last HST\_N solutions; it is sufficient to store the makespan and sequencing list pointers to characterize a solution. HST\_N should be small with respect to memory requirements and time required to check for cycling, but sufficiently large to avoid missing cycles having a moderately long period. For the test problems, a HST\_N of 50 was found to be a reasonable value on average.

Each iteration, after a best move has been identified and performed, the resulting



solution is added to the history list. A simple heuristic test is then performed to detect possible cycling behavior. A *sliding window* of prespecified width  $w$  scans the history list for a pattern of makespans matching the most recent  $w$  solutions. A match triggers an explicit comparison of the recent and older solutions; a positive comparison is taken as indicative of probable cycling. The window width must be large enough to provide a reasonably accurate test, but small enough that comparison times are modest. A window width of  $w=3$  was found to be effective.

If this test detects a repeated solution pattern, a cycling strategy is invoked. In experiments in which cycling simply triggered diversification using the solution stack, the stack was exhausted too quickly to allow adequate exploration of neighboring solutions. We make the assumption that the search is most probably cycling because the short term memory has been allowed to take on values which are too small for the current region of exploration. We combat this by incrementing the lower limit on short term memory, *stm\_lo*, for the duration of the current search. All tabu information is left intact, and the working value of short term memory is set to *stm\_hi* in order to encourage a rapid departure from the area of cycling. Obviously, the history list must be cleared to prevent spurious re-detection of the remembered solution pattern. If cycling in the current search has been so pronounced as to have already driven *stm\_lo* up to *stm\_hi*, we abandon the region by invoking the diversification strategy (see Figure 5).

#### *Move selection*

Machines are scanned successively for critical operation pairs meeting the criteria described above. The effect of exchanging each such pair is evaluated using Balas'  $\Delta$ . Move comparison is "strictly-less-than": the first move yielding a specific improvement (*i.e.*, move value) is chosen.

However, the move evaluation procedure may also report a "no-moves" situation, that is, that there is no move available which is either nontabu or which satisfies the aspiration criterion. Several strategies for dealing with this situation were investigated.

We have previously combatted a no-moves condition by simply clearing the tabu information and continuing from the current solution. Unfortunately, tracing studies suggest that this approach may unnecessarily discard potentially useful information about the region, leading in some cases to cycling and an eventual departure *via* the diversification strategy, with no additional exploration having been performed.

Another approach, constructed by symmetry with the cycling strategy, is to assume that potential candidate moves are all tabu because the length of short term memory is too large. Therefore, the upper limit on short term memory is decreased for the duration of the current search, and the working value of short term memory is set to the new upper limit, in order to encourage departure from the current region. If the tabu information is not cleared, the upper limit on short term memory (and the working short term memory length set from it) must be reduced sufficiently to allow at least one move to become nontabu. This implicit use of the oldest tabu move, in conjunction with short term memory adjustment, was not competitive with a more straightforward use of the oldest tabu move discussed below. On the other hand, if the tabu information is cleared, a situation may quickly develop in which the cycling and no-moves strategies alternate as the upper and lower limits on short term memory length converge, until the cycling strategy triggers the diversification strategy.

Finally, one may simply abandon the no-moves concept entirely. In such a case, one might (a) use the tabu move most improving relative to the current search (a localized or pseudo-aspiration criterion); (b) randomly select a tabu move; or (c) use the oldest tabu move, making no other changes. Empirically, this last strategy proved best and is implemented in the current study.

#### *Search control*

Initial values of *stm\_lo* and *stm\_hi* are prespecified. The search commences from the best dispatching solution, using a working short term memory length of *stm\_hi*. At each iteration, a best move is determined and executed, and the resulting solution is added to the solution history. If a cycle is detected, the cycling strategy is performed, else the improving or nonimproving move strategy is performed, depending on the value of the executed move. The search is terminated when the allotted CPU time is exceeded, the *a priori* lower bound is achieved, or the solution stack is exhausted. The high level search strategy is sketched in Figure 6.

### **4. Computational Experience**

In order to measure the performance of the adaptive, dynamic tabu search strategy described above, a set of test runs was performed using a selection of the classical job shop problems we considered previously (see, e.g., Barnes and Chambers [1995]), specifically, MT10 [Fisher and Thompson 1963], LA1-40 [Lawrence 1984], and ABZ5-9 [Adams, Balas, and Zawack 1988]. The method was implemented in C, and run in single user mode on a DEC 3000 Model 600 AXP.

The initial values of *stm\_lo* and *stm\_hi* were successively selected in the interval from 3 to 20 inclusive, with exploration of each short term memory range being limited to 120 seconds. Ranges for which *stm\_lo* and *stm\_hi* were either both relatively small or both relatively large led to search termination far earlier than this. Tracing studies indicate that small values allow cycling behavior to develop, exhausting the solution stack quickly. Conversely, larger values tend to discourage adequate localized exploration; the limit on nonimproving moves is soon exceeded, again leading to exhaustion of the solution stack and termination of the search. Except where noted, a nonimproving moves limit of 2000 moves was used.

As in our earlier investigations, we have avoided attempts to construct artificial scaling factors to compare results reported by other authors using different computer platforms. However, for purposes of reference, the test run of MT10 was performed on several different machines. Table 1 summarizes these results.

The first column of Table 1 indicates a selected short term memory range, i.e., values of *stm\_lo* and *stm\_hi*. The second column indicates the time required to first achieve the known optimum of 930 using the short term memory range in the first column. Finally, the third column indicates the corresponding computer platform (as listed in the footnote). A short term memory range of (8,9) performed well for this problem; however, several other ranges found a solution of 930, given sufficient time, and are noted in the table also. Further, it was observed that the nonimproving moves limit of 2000 was much larger than necessary for this problem, wasting a significant amount of search time. Adjustment of this limit to 700 for the short term memory range

(8,9) reduced the time to best by more than 30%, to a value of 3.52 seconds on the DEC 3000 platform.

Tables 2 - 4 summarize computational results for the remainder of the problems. The second column lists two numbers, the trivial *a priori* lower bound described in Section 3, and the makespan of the best dispatching solution used as the initial solution. The third column is the best result for our earlier tabu search strategy. The upper number is makespan (optimal solutions are shown in boldface), and the lower number (in italics) is the "time to best".

The next column gives the results reported by Dell'Amico and Trubian [1993] for their adaptive, dynamic tabu search. (Unfortunately, Taillard [1989, 1994] does not report sufficient information for these test problems to warrant comparison.) Due to the randomized nature of their method, the authors performed five runs from different starting solutions for each problem. They report average and best makespan, and average and maximum time, but not time to best. In our tables, the figures in the fourth column are Dell'Amico and Trubian's best makespan and average time.

The fifth column gives the best makespan and time to best achieved by the tabu search method described in Section 3 for any short term memory range investigated, using a limit of 120 seconds per range and a nonimproving moves limit of 2000. Finally, the sixth column gives the optimum, or best bounds, as furnished by Applegate and Cook.

Table 2 presents the results for the 10-machine 10-job problems. For LA16-20 and ABZ6, our dynamic tabu search is competitive with both our previous semidynamic method and with Dell'Amico and Trubian. Dynamic tabu search achieved the optimum for ABZ5, and also did so more efficiently for MT10 than did our earlier technique.

Table 3 presents the results for the rectangular 10-machine problems. In every instance, dynamic tabu search was equal or superior to our previous method (except for a special run of LA21). It was competitive on six problems with Dell'Amico and Trubian, and was superior on four, achieving the optimum in two cases (LA22 and LA25), and significantly improving the best upper bound in one (LA29).

Finally, Table 4 presents the results for the 15-machine, 15- and 20-job problems. In all but ABZ9 (and one special run of ABZ7), dynamic tabu search was again superior to our semidynamic method. It was also superior in six out of eight cases to Dell'Amico and Trubian, achieving the optimum in two of these (LA36 and LA39). We expect that improved performance would be realized in ABZ7 and ABZ9 by carefully tuning the limit on nonimproving moves.

## 5. Directions for Future Research

Recent investigations [Battiti and Tecchiolli 1994; Battiti 1995; Barnes and Carlton 1995; Carlton 1995] suggest that it may be possible to construct a dynamic TS approach which is fully adaptive to changing search conditions without the need for extensive parameterization or tuning. Such an implementation for JSSP would be of interest.

Additionally, significant attention has been focused on flexible manufacturing systems (or on flexible job shops as an approximation), in which more than one machine may be available to process a particular type of operation. We are investigating concurrent multi-neighborhood TS strategies for such problems.



## Bibliography

- ADAMS, J., E. BALAS, AND D. ZAWACK. 1988. The shifting bottleneck procedure for job shop scheduling. *Management Science*, **34**:3, 391-401.
- APPLEGATE, D. AND W. COOK. 1991. A computational study of the job-shop scheduling problem. *ORSA Journal on Computing*, **3**:2, 149-156.
- BAKER, K.R. 1974. *Introduction to Sequencing and Scheduling*. John Wiley & Sons, New York.
- BALAS, E. 1969. Machine sequencing via disjunctive graphs: An implicit enumeration algorithm. *Operations Research*, **17**, 941-957.
- BARNES, J.W. AND W.B. CARLTON. 1995. *Solving the vehicle routing problem with time windows using reactive tabu search*. INFORMS conference, New Orleans LA, Oct 1995.
- BARNES, J.W. AND J.B. CHAMBERS. 1991a. Solving the job shop scheduling problem using tabu search. *Technical Report Series ORP 91-06*, Graduate Program in Operations Research and Industrial Engineering, Department of Mechanical Engineering, The University of Texas at Austin.
- BARNES, J.W. AND J.B. CHAMBERS. 1991b. Solving the job shop scheduling problem using tabu search. ORSA/TIMS joint meeting, Anaheim CA, Nov 1991.
- BARNES, J.W. AND J.B. CHAMBERS. 1992a. Additional results on solving the job shop scheduling problem using tabu search. ORSA/TIMS CS Technical Section meeting, Williamsburg VA, Jan 1992.
- BARNES, J.W. AND J.B. CHAMBERS. 1992b. Historical generators and ejection chains in tabu search for the job shop scheduling problem. ORSA/TIMS joint meeting, San Francisco CA, Nov 1992.
- BARNES, J.W. AND J.B. CHAMBERS. 1995. Solving the job shop scheduling problem using tabu search. *IIE Transactions*, **27**, 257-263.
- BARNES, J.W. AND M. LAGUNA. 1991. A review and synthesis of applications of tabu search to production scheduling problems. *Technical Report Series ORP 91-05*, Graduate Program in Operations Research and Industrial Engineering, Department of Mechanical Engineering, The University of Texas at Austin.
- BATTITI, R. 1995. Reactive search: toward self-tuning heuristics. Keynote talk at Applied Decision Technologies, Brunel UK, April 3-4, 1995.
- BATTITI, R. AND G. TECCHIOILLI. 1994. The reactive tabu search. *ORSA Journal on Computing*, **6**:2, 126-140.
- CARLTON, W.B. 1995. *A tabu search approach to the general vehicle routing problem*. Ph.D. dissertation, The University of Texas at Austin.
- DELL'AMICO, M., AND M. TRUBIAN. 1993. Applying tabu search to the job-shop scheduling problem. *Annals of Operations Research*, **41**, 231-252.
- FISHER, H., AND G.L. THOMPSON. 1963. Probabilistic learning combinations of local job-shop scheduling rules. In *Industrial Scheduling*, J.F. Muth and G.L. Thompson (eds.), Prentice-Hall, Englewood Cliffs NJ. 225-251.
- FRENCH, S. 1982. *Sequencing and Scheduling: An Introduction to the Mathematics of the Job-Shop*. Halsted Press (John Wiley & Sons), New York.
- GAREY, M.R., AND D.S. JOHNSON. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman, New York.
- GAREY, M.R., D.S. JOHNSON, AND R. SETHI. 1976. The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, **1**, 117-129.
- GLOVER, F. 1989. Tabu search: Part I. *ORSA Journal on Computing*, **1**:3, 190-206.
- GLOVER, F. 1990. Tabu search: Part II. *ORSA Journal on Computing*, **2**:1, 4-32.
- GLOVER, F. 1991. Tabu search for nonlinear and parametric optimization (with links to genetic algorithms). *Discrete Applied Mathematics* (to appear).
- GLOVER, F. 1994. Tabu search: New options for optimization. *ORSA CSTS Newsletter*, **15**:2, 1 and 13-20.
- GLOVER, F., AND M. LAGUNA. 1993. Tabu search. In *Modern Heuristic Techniques for Combinatorial Problems*, C.R. Reeves (ed.), Blackwell, Oxford. 70-150.
- GLOVER, F., E. TAILLARD, AND D. DEWERRA. 1993. A user's guide to tabu search. *Annals of Operations Research*, **41**, 3-28.

- LAWRENCE, S. 1984. Resource constrained project scheduling: An experimental investigation of heuristic scheduling techniques. Graduate School of Industrial Administration, Carnegie-Mellon University.
- MATSUO, H., C.J. SUH, AND R.S. SULLIVAN. 1988. A controlled search simulated annealing method for the general jobshop scheduling problem. *Working paper 03-04-88*, Department of Management, Graduate School of Business, The University of Texas at Austin.
- NOWICKI, E., AND C. SMUTNICKI. 1993. A fast taboo search algorithm for the job shop problem. *Preprinty nr 8/93*, Instytut Cybernetyki Technicznej, Politechniki Wrocławskiej, Wrocław.
- ROY, B., AND B. SUSSMAN. 1964. Les problèmes d'ordonnancement avec contraintes disjonctives. *Note DS No 9 bis*, SEMA, Paris.
- SEEDGEWICK, R. 1983. *Algorithms*. Addison-Wesley, Reading MA.
- SUH, C.J. 1988. *Controlled Search Simulated Annealing for Job Scheduling*. Ph.D. dissertation, The University of Texas at Austin.
- TAILLARD, E. 1989 (revised 1992). Parallel taboo search techniques for the job shop scheduling problem. *Report ORWP 89/11*, Département de mathématiques, École Polytechnique Fédérale de Lausanne.
- TAILLARD, E. 1994. Parallel taboo search techniques for the job shop scheduling problem. *ORSA Journal on Computing*, 6:2, 108-117.
- VAESSENS, R.J.M., E.H.L. AARTS, AND J.K. LENSTRA. 1994. Job shop scheduling by local search. *Memorandum COSOR 94-05*, Department of Mathematics and Computing Science, Eindhoven University of Technology.
- VAN LAARHOVEN, P.J.M., E.H.L. AARTS, AND J.K. LENSTRA. 1992. Job shop scheduling by simulated annealing. *Operations Research*, 40:1, 113-125.

---

```
move evaluation loop {  
    ...  
    save operation times;  
    switch ( move_type ) {  
        ...  
        case EXCHANGE:  
            if ( Balas  $\Delta$   $\geq$  0 ) use as move value;  
            else compute makespan in forward pass;  
            break;  
        ...  
    }  
    restore operation times;  
    ...  
}  
...  
select and perform best admissible move;  
perform sort and forward pass yielding new makespan;  
perform backward pass yielding values for new Balas  $\Delta$ s;  
...
```

---

**Figure 1. Move evaluation**

---

```
// stm      working value of short term memory
// stm_lo   current minimum short term memory
// stm_hi   current maximum short term memory

adjust short term memory {
    if ( move value < 0 ) {
        if ( stm > stm_lo ) stm -= 1;
    }
    else {
        if ( stm < stm_hi ) stm += 1;
    }
}
```

---

**Figure 2. Short term memory adjustment**

---

```
// Z_bst    best makespan seen to date
// Z_cur    makespan of current solution
// Z_ref    reference makespan for current search phase
// Ni_n     count of consecutive nonimproving moves,
//          relative to current search

improving move strategy {
    Z_ref = Z_cur;
    Ni_n = 0;
    adjust short term memory;
    if ( Z_cur < Z_bst ) {
        Z_bst = Z_cur;
        push current solution on solution stack;
    }
}
```

---

**Figure 3. Improving move strategy**



---

```
// stm      working value of short term memory
// stm_lo   current minimum short term memory,
//           initially set to stm_lo_orig
// stm_hi   current maximum short term memory,
//           initially set to stm_hi_orig
// Ni_n     count of consecutive nonimproving moves,
//           relative to current search
// Ni_max   prespecified limit on Ni_n

diversification strategy {
    if ( solution stack is empty ) STOP;
    pop solution stack and install as current solution;
    stm_lo = stm_lo_orig;
    stm_hi = stm_hi_orig;
    stm = stm_hi;
    Ni_n = 0;
    clear tabu information;
    clear history list;
}

nonimproving move strategy {
    if ( ++Ni_n > Ni_max ) perform diversification strategy;
    else adjust short term memory;
}
```

---

**Figure 4. Diversification and nonimproving move strategy**

---

```
// stm      working value of short term memory
// stm_lo   current minimum short term memory
// stm_hi   current maximum short term memory

cycling strategy {
    if ( stm_lo >= stm_hi ) {
        perform diversification strategy;
    }
    else {
        stm_lo += 1;
        stm = stm_hi;
        clear history list;
    }
}
```

---

**Figure 5. Cycling strategy**

---

```

// stm      working value of short term memory
// stm_lo   current minimum short term memory, initially stm_lo_orig
// stm_hi   current maximum short term memory, initially stm_hi_orig
// Ni_n     count of consecutive nonimproving moves in current search,
//          limited to Ni_max

input problem model, max cpu time, Ni_max, stm_lo_orig, stm_hi_orig;

select best dispatching solution and install as starting solution;

search {
  initialization:
    stm_lo = stm_lo_orig; stm_hi = stm_hi_orig; stm = stm_hi;
    initialize history list; initialize tabu information;
    Ni_n = 0;

  while ( cpu time not exceeded ) {
    select best move;
    perform best move {
      execute move;
      recompute cpm information;
      if ( lower bound achieved ) {
        stop with optimum;
      }
      add solution to history list;
    }
    perform cycle check;
    if ( cycle detected ) {
      perform cycling strategy;
    }
    else if ( new makespan improves current search ) {
      perform improving move strategy;
    }
    else {
      perform nonimproving move strategy;
    }
  }
}

```

---

**Figure 6. Tabu search strategy**

**Table 1**  
**Sample results for the Fisher-Thompson 10x10 problem [1963]**

Short term memory range	Time to best (Z = 930)	Platform
Ni_max = 2000		
8,9	23.60	A
8,9	5.12	B
3,20	6.97	
4,14	29.88	
4,12	58.21	
Ni_max = 700		
8,9	3.52	B
	12.15	C
	20.42	D
	25.55	E

- 
- A) IBM RS 6000/350
  - B) DEC 3000/600 (AXP)
  - C) DEC 5000/200 (MIPS)
  - D) SUN 4/50 SPARCStation IPX  
(SUN ANSI C 2.0.1)
  - E) i486 DX2 50 MHz  
(WATCOM C/32 9.5(b))

**Table 2**  
**Results for 10-machine 10-job problems**

<b>Problem</b>	<b>(Bound, Dispatch)</b>	<b>B &amp; C<sup>a,b</sup></b>	<b>D &amp; T<sup>c</sup></b>	<b>Dynamic<sup>d,e</sup> Tabu Search (120s)</b>	<b>A &amp; C<sup>f</sup> Value</b>
LA16	(717,1054)	<b>945</b> 187.5	<b>945</b> 97.4	<b>945</b> 9.75	<b>945</b>
LA17	(683,846)	<b>784</b> 23.1	<b>784</b> 21.7	<b>784</b> 0.52	<b>784</b>
LA18	(663,907)	<b>848</b> 11.8	<b>848</b> 63.1	<b>848</b> 0.08	<b>848</b>
LA19	(685,940)	843 217.1	<b>842</b> 103.8	<b>842</b> 0.40	<b>842</b> 1462.3
LA20	(756,964)	<b>902</b> 30.7	<b>902</b> 71.7	<b>902</b> 0.67	<b>902</b> 1402.3
ABZ5	(868,1351)	1238 75.4	1236 139.5	<b>1234</b> 55.65	<b>1234</b> 951.5
ABZ6	(742,981)	<b>943</b> 62.5	<b>943</b> 86.8	<b>943</b> 5.17	<b>943</b> 90.9
MT10	(655,1007)	<b>930</b> 215.8	935 155.8	<b>930</b> 5.12	<b>930</b> 372.4

- 
- a) Max nonimproving moves set at 5000  
b) IBM RS 6000  
c) PC 386 33 MHz  
d) Max nonimproving moves set at 2000  
e) DEC 3000/600 (AXP)  
f) SUN SPARC Station 1



**Table 3**  
**Results for rectangular 10-machine problems**

Problem	(Bound, Dispatch)	B & C <sup>a,b</sup>	D & T <sup>c</sup>	Dynamic <sup>d,e</sup> Tabu Search (120s)	A & C <sup>f</sup> Value
<b>15-job problems</b>					
LA21	(935,1253)	1050 <sup>g</sup> 166.0	1048 198.8	1048 52.81	(1040,1053)
LA22	(830,1058)	932 220.5	933 191.4	<b>927</b> 40.65	<b>927</b>
LA23	( <b>1032</b> ,1124)	<b>1032</b> 2.7	<b>1032</b> 1.8	<b>1032</b> 0.28	<b>1032</b>
LA24	(857,1041)	946 27.7	941 181.8	939 26.28	<b>935</b>
LA25	(864,1099)	988 176.1	979 191.7	<b>977</b> 12.85	<b>977</b>
<b>20-job problems</b>					
LA26	( <b>1218</b> ,1435)	<b>1218</b> 30.5	<b>1218</b> 22.1	<b>1218</b> 1.58	<b>1218</b>
LA27	(1188,1369)	1250 161.8	1242 254.2	1242 83.65	(1235,1269)
LA28	( <b>1216</b> ,1391)	1225 165.9	<b>1216</b> 186.4	<b>1216</b> 17.37	<b>1216</b>
LA29	(1105,1337)	1194 <sup>h</sup> 195.6	1182 281.3	1168 57.08	(1120,1195)
LA30	( <b>1355</b> ,1427)	<b>1355</b> 2.2	<b>1355</b> 10.4	<b>1355</b> 0.30	<b>1355</b>

- 
- a) Max nonimproving moves set at 5000  
 b) IBM RS 6000  
 c) PC 386 33 MHz  
 d) Max nonimproving moves set at 2000  
 e) DEC 3000/600 (AXP)  
 f) SUN SPARC Station 1  
 g) la21 value of 1047 obtained by 11-13 short term memory range in 318.2s  
 h) la29 value of 1191 obtained with special short term memory range

**Table 4**  
**Results for 15-machine problems**

Problem	(Bound, Dispatch)	B & C <sup>a,b</sup>	D & T <sup>c</sup>	Dynamic <sup>d,e</sup> Tabu Search (120s)	A & C <sup>f</sup> Value
<b>15-job problems</b>					
LA36	(1028,1433)	1278 221.4	1278 238.4	<b>1268</b> 76.96	<b>1268</b>
LA37	(986,1588)	1418 232.0	1409 242.2	1407 6.47	<b>1397</b>
LA38	(943,1407)	1211 180.9	1203 256.6	1202 31.30	(1184,1217)
LA39	(1012,1381)	1237 258.7	1242 237.8	<b>1233</b> 20.92	<b>1233</b>
LA40	(1027,1424)	1228 93.9	1233 236.6	1226 119.41	<b>1222</b>
<b>20-job problems</b>					
ABZ7	(556,745)	674 <sup>g</sup> 339.0	667 320.1	668 45.50	(654,668)
ABZ8	(566,798)	682 296.3	678 336.1	676 49.58	(635,687)
ABZ9	(563,826)	693 393.0	692 320.8	694 43.46	(656,707)

- 
- a) Max nonimproving moves set at 6000  
 b) IBM RS 6000  
 c) PC 386 33 MHz  
 d) Max nonimproving moves set at 2000  
 e) DEC 3000/600 (AXP)  
 f) SUN SPARC Station 1  
 g) abz7 makespan of 668 obtained by 15-15 short term memory range in 374.7s