CHAPTER 5

JOB SHOP SCHEDULING USING FUZZY WITH TABU SEARCH

This chapter illustrates TS meta-heuristic approach to solving JSSP. Improvement of the constructed solution using the proposed approach is presented. Different procedures for finding neighborhood solutions are evaluated, and the best one is found. New dynamic tabu length method, which improves the quality of the solution, is illustrated.
5.1 INTRODUCTION

Glover (1989, 1990) presented the fundamental principles of Tabu Search (TS) as a strategy for combinatorial optimization problems. Tabu search is a meta-heuristic designed for finding a near optimal solution of combinatorial optimization problems. TS provide solutions very close to optimality and used to tackle difficult problems. These successes have made TS extremely popular among those interested in finding viable solutions to the large combinatorial problems encountered in many practical settings. TS based on Local Search (LS) improvement techniques. LS can be roughly summarized as an iterative search procedure that, starting from an initial feasible solution, progressively
improves it by applying a series of local modifications or moves. At each iteration, the search moves to an improved feasible solution that differs slightly from the current one. In LS, the quality of the solution obtained and computing times are usually highly dependent upon the “richness” the set of transformations considered at each iteration of the heuristic.

In the majority of industrial settings, different types of job compete with one another for the available scarce resources. A schedule is a feasible resolution of the resource constraints when no operation has ever occupied the same machine simultaneously. Proper scheduling of jobs is very important for the successful operation of a shop. Much research has been carried
out in the scheduling area where approaches have ranged from the use of a Gantt chart to optimization methods such as dynamic programming, integer programming and branch and bound methods. However, most scheduling problems are classified as NP-complete, and optimum seeking methods have been found to be impractical for a real-size manufacturing shop. As mentioned earlier, this leads to the development of heuristic techniques such as neighbourhood search and random sampling. In recent years, journals in a wide variety of fields have published tutorial articles and computational studies documenting successes by tabu search in extending the frontier of problems that can be handled effectively yielding solutions whose quality
often significantly surpasses that obtained by methods previously applied.

5.2 ELEMENTS OF TABU SEARCH

5.2.1 Tabu List

The purpose of the tabu list is to prevent the search from degenerating by starting to cycle between the same solutions. In a tabu list, the elements added to the list are attributing to save computer memory. The tabu list starts with empty. The length of tabu list is important for selection of the best solution.

5.2.2 Short-Term Memory

The kind of memory that is limited in terms of time and storage
capacity. In TS, the tabu list can be regarded as a short-term memory. Regency memory which will be defined later is also a short-term memory. With a short term memory, a revisited solution may be revisited with a different neighborhood.

5.2.3 Neighborhood

A neighborhood structure is a mechanism, which contains a new set of neighbors solution by applying a simple modification to a given solution. Each neighbour solution is reached from a given solution by the move. A sequencing move is defined by the exchange of certain adjacent critical operation pairs within the block and then considered the exchange of every adjacent critical operation pair on every machine.
5.2.4 Tabu Move

Tabu move is used to determine whether a solution with a characteristic attribute has been visited or not. If TS finds a candidate solution that possesses the attributes of a recently visited solution within tabu tenure, the move is forbidden, and the next candidate move is entertained.

5.2.5 Tabu Tenure

Tabu tenure is used to force the search process to visit the search regions that are not yet explored. For each swapped move which job i in position l and job k in position r have exchanged, the tabu tenure is penalized according to the frequency of job i in position r and the frequency of job k in position l. For each inserted move which the
job in position 1 is inserted into position r, r>1, the tabu tenure is penalized according to the frequency of job i in position k where r ≤ k ≤ l.

5.2.6 Aspiration Criteria

An aspiration criterion is defined to deal with the case in which an interesting move is tabu. If a current tabu move satisfies the aspiration criterion, its tabu status is cancelled, and it becomes an allowable move. In order to improve the performance of TS, intensification strategies can be used to accentuate the search in a promising region of the solution space. An intensification strategy consists of storing the elite solutions and restarts the search for them so as to explore neighborhoods with potentially good solutions. The recovery of
the elite solutions is deferred until the last stage of the search. The elite solutions, a set of best solutions found so far, are recovered in the list of the worst solution to the best solution.

5.3 SOLUTION REPRESENTATION

Algorithm TS_Procedure( ) (table 5.1) gives steps for the proposed fuzzy with tabu search method, and this algorithm employs different neighborhood and dynamic tabu length strategies, which are described in the next two sections, to improve the initial solution. Different parameters are given to the algorithm. MAXCYCNO and IL represent the total number of iterations and initial length of the tabu list respectively. MAXT represents the maximum number of times, for which the
improvement is not made during the construction of solutions. Initial solution $S^*$ is constructed using fuzzy approach information. The algorithm finds a set of neighbors for the current solution and tabu length for the current iteration. If the selected neighbor $s_i$ ($0 \leq i \leq k$) (Eswaramoorthy et al., 2007) is not in tabu or an aspiration criterion is met, the neighbor $s_i$ is added to tabu.

**Table 5.1 Algorithm 1: TS_Procedure**

<table>
<thead>
<tr>
<th>Algorithm :TS_Procedure()</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Step 1:</strong></td>
</tr>
<tr>
<td>Assign 0 to CYCNO, TP, T;</td>
</tr>
<tr>
<td>Initialize MAXCYCNO, MAXT, IL</td>
</tr>
<tr>
<td>Assign IL to TL</td>
</tr>
<tr>
<td>Generate initial solution $S^*$ using fuzzy sequence scheduler</td>
</tr>
<tr>
<td>Set makespan for $S^<em>$ to $f(S^</em>)$</td>
</tr>
</tbody>
</table>
**Table 5.1 (Continued)**

<table>
<thead>
<tr>
<th>Step 2:</th>
</tr>
</thead>
<tbody>
<tr>
<td>If TP &gt; TL Then</td>
</tr>
<tr>
<td>Assign 0 to TP</td>
</tr>
<tr>
<td>EndIf</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Step 3:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add = False</td>
</tr>
</tbody>
</table>

Let N(S) be the set of neighbors for the current iteration

Find N’(S) as \{s1,s2,…..sk\} from N(S) by using neighborhood strategy

Sort elements in N’(S) according to solution quality in ascending order

For i = 0 to k do

If si is in not in Tabu Then

Add = True

Else If Aspiration Criterion is met for siThen

Add = True

EndIf

EndIf

If Add = True Then

Add si to Tabu in position TP

Break

EndIf
Table 5.1 (Continued)

If Add = False Then
    Find a neighbor si which is old in Tabu
    Add si to Tabu in position TP
EndIf

Step 4:

    Apply neighbor from Tabu in position TP
    Find the current solution S
    If f(S) < f(S*) Then  //Save the best so far solution
        Assign S to S*, f(S) to f(S*), and 0 to T
    EndIf
    Increment CYCNO, T and TP by 1

Step 5:

    If CYCNO > MAXCYCNO or T > MAXT Then
        Go to Step 6
    Else
        Go to Step 2
    EndIf

Step 6:

    Print S* and f(S*)
    Stop
The aspiration criterion is used to check the condition $f(S) < f(S^*)$, where $f(S)$ is the makespan of the neighborhood solution $S$ produced by the application of a neighbor $s_i$, which is already in tabu. If the neighbor can not be added to tabu, old neighbor from the set $N(S)$ is added to tabu.

The old neighbor means the earliest time, in which a neighbor among the set of neighbors $N(S)$ was added to tabu. This process is repeated until a termination criterion is met. The termination criterion is either reaching the maximum number of iterations or a pre-specified threshold value or a fixed amount of CPU time or no improvement of the constructed solution for MAXT number of iterations.
5.4 GENERATION OF NEIGHBORHOOD

Let N(S) be a set of neighbors for the current solution S. Blocks are constructed from the N(S) and different neighborhood strategies are used to select the appropriate set of neighbors N’(S) for each iteration. Blocks are found for each machine. The block is successive operations in the critical path pertaining to a particular machine. A data structure numblock[i] is employed to represent the total number of blocks in \( i^{th} \) machine and is implemented as an integer vector containing \( m \) locations. Two more data structures block[i][j], node[k] and block[i][j]. count are implemented as an integer vector and an integer, and hold the position of the \( k^{th} \) node in \( j^{th} \) block of \( i^{th} \) machine and total number
of nodes in \( j^{th} \) block of \( i^{th} \) machine respectively. The block is defined as follows:

\[
\text{struct\{int count; int nodes[ ];\} block[ ][ ];}
\]

Size of vectors block and nodes depend upon the problems that are used to solve. A structure called neighbor list is defined as a two-dimensional array, which holds a set of neighbors for the current iteration and is given as follows:

\[
\text{struct\{ int start; int end;\} neighborlist[ ][ ];}
\]

Two integer elements neighborlist[i].start and neighborlist[i].end represent the start and end nodes of \( i^{th} \) neighbor respectively. The size of the vector neighbor list also depends upon the size of
the problem. Four variations of the neighborhood strategy are discussed in this section.

5.5 IMPLEMENTING FIRST AND LAST NEIGHBOR ALGORITHM

An algorithm for FLN strategy is represented in Table 5.2:

**Table 5.2 Algorithm 2: First_Last_Neighbor()**

<table>
<thead>
<tr>
<th>Algorithm: First_Last_Neighbors( )</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Inputs</strong></td>
</tr>
<tr>
<td>numblock[i] – number of blocks of ( i^{th} ) machine</td>
</tr>
<tr>
<td>block[i][j] – ( j^{th} ) block of ( i^{th} ) machine</td>
</tr>
<tr>
<td>block[i][j].count – number of nodes in ( j^{th} ) block of ( i^{th} ) machine</td>
</tr>
<tr>
<td>block[i][j].node[k] – ( k^{th} ) operation in ( j^{th} ) block of ( i^{th} ) machine</td>
</tr>
<tr>
<td><strong>Output</strong></td>
</tr>
<tr>
<td>( N'(S) ) – Set having first and last neighbors in each block</td>
</tr>
</tbody>
</table>
Table 5.2 (Continued)

```
Begin
    Let N’(S) = φ, Initialize k by 1
    For i = 1 to m do
        For j = 1 to numbloc[k] do
            Let strat[k] = block[i][j].node[1]
            Let end[k] = block[i][j].node[2]
            Add (strat[k], end[k]) to N’(S),
            Increment k by 1
            If block[i][j].count > 2 Then
                //Check for more neighbor
                Let start[k] = block[i][j].node[block[i][j].count-1]
                Let end[k] = block[i][j].node[block[i][j].count]
                Add (start[k], end[k]) to N’(S),
                Increment k by 1
            EndIf
        EndFor
    EndFor
End
```

Procedure First_Last_Neighbors( ) selects only first two and last two nodes in each
block. For example, if a block contains a set of nodes \{1, 2, 3, 4, 5\}, \(N'(S)\) contains a set of neighbors \{(1, 2), (4, 5)\}. If the number of nodes in a block are less than or equal to three, all nodes are considered for the construction of neighbors. For example, if a block contains a set of nodes \{1, 2, 3\}, \(N'(S)\) contains a set of neighbors \{(1, 2), (2, 3)\}.

### 5.6 IMPLEMENTING MIDDLE NEIGHBOR ALGORITHM

An algorithm for MN strategy is represented Table 5.3:
### Table 5.3 Algorithm 3: Middle Neighbor()

**Algorithm:** Middle_Neighbors( )

**Inputs:**
- `numblock[i]` – number of blocks in `ith` machine
- `block[i][j]` – `jth` block of `ith` machine
- `block[i][j].count` – number of nodes in `jth` block of `ith` machine
- `block[i][j].node[k]` – `kth` operation in `jth` block of `ith` machine

**Output:**
- `N'(S)` – Set having middle neighbors in each block

**Begin**

Let `N'(S) = φ`

Initialize `k` by 1

For `i = 1` to `m` do

For `j = 1` to `numblock[i]` do

If `block[i][j].count <= 3` Then

/*Find first neighbor from first two nodes in case of total nodes in the block is equal to two*/

Let `start[k] = block[i][j].node[1]`

Let `end[k] = block[i][j].node[2]`

Add (`start[k]`, `end[k]`) to `N'(S)`, Increment `k` by 1

**End**
/*Find second neighbor from last two nodes in case to total nodes in the block is equal to three*/

If block[i][j].count = 3 Then
Let start[k] = block[i][j].node[2]
Let end[k] = block[i][j].node[3]
Add (start[k], end[k]) to N'(S), Increment k by 1
EndIf
Else
/*Find set of neighbors by omitting first and last nodes in the block in case of total nodes in the block is greater than three*/

For p = 2 to block[i][j].count – 1 do
Let start[k] = block[i][j].node[p]
Let end[k] = block[i][j].node[p+1]
Add (start[k], end[k]) to N'(S), Increment k by 1
EndFor
EndIf
EndFor
EndFor
End
Procedure Middle_Neighbors(
) selects only middle nodes (omits first and last nodes) in each block. For example, if a block contains a set of nodes \{1, 2, 3, 4, 5\}, N'(S) contains a set of neighbors \{(2, 3), (3, 4)\}. If number of nodes in a block are less than or equal to three, all nodes are considered for the construction of neighbors. For example, if a block contains a set of nodes \{1, 2, 3\}, N'(S) contains a set of neighbors \{(1, 2), (2, 3)\}.

5.7 IMPLEMENTING ALL NEIGHBOR ALGORITHM

Algorithm for ALL strategy is represented Table 5.4:
Table 5.4 Algorithm 4: All_Neighbors()

**Algorithm : All_Neighbors()**

**Inputs:**
- `numblock[i]` – number of blocks in i\(^{th}\) machine
- `block[i][j]` – j\(^{th}\) block of i\(^{th}\) machine
- `block[i][j].count` – number of nodes in j\(^{th}\) block of i\(^{th}\) machine
- `block[i][j].node[k]` – k\(^{th}\) operation in j\(^{th}\) block of i\(^{th}\) machine

**Output:**
- N\('\)\(S\) – Set having all neighbors in each block

**Begin**

Let N\('\)\(S\) = φ

Initialize k by 1

For i = 1 to m do

For j = 1 to `numblock[i]` do

For p = 1 to `block[i][j].count-1` do

Let start[k] = `block[i][j].node[p]`

Let end[k] = `block[i][j].node[p+1]`

Add (start[k], end[k]) to N\('\)\(S\)

Increment k by 1

EndFor EndFor EndFor

End
Procedure All_Neighbors()
selects all neighbors in each block. For example, if a block contains a set of nodes 
\{1, 2, 3, 4\}, \( N'(S) \) contains a set of neighbors \( \{(1, 2), (2, 3), (3, 4)\} \).

5.8 IMPLEMENTING NEIGHBORS
WITH CYCLES ALGORITHM

An algorithm for NWC strategy is given below:

Table 5.5 Algorithm 5: Neighbors_with_Cycles()

<table>
<thead>
<tr>
<th>Algorithm : Neighbors-With_Cycles()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inputs:</td>
</tr>
<tr>
<td>CYCNO – current iteration number</td>
</tr>
<tr>
<td>( v0 ) – variation parameter (The value between 0 and 10)</td>
</tr>
<tr>
<td>numblock[i] – number of blocks in ( i^{\text{th}} ) machine</td>
</tr>
<tr>
<td>block[i][j]\ – ( j^{\text{th}} ) block of ( i^{\text{th}} ) machine</td>
</tr>
</tbody>
</table>
Table 5.5 (Continued)

| block[i][j].count – number of nodes in j<sup>th</sup> block of i<sup>th</sup> machine |
| block[i][j].node[k] – k<sup>th</sup> operation in j<sup>th</sup> block of i<sup>th</sup> machine |

Output:
N’(S) – Set having neighbors in each block according to iteration number and variation parameter

Begin

Get CYCNO, V, Let N’(S) = φ

If CYCNO % v0 = 0 Then

Call All_neighbors( )

Else

Initialize k by 1

For i = 1 to m

For j = 1 to numb[1] do

If block[i][j].count <3 Then

Let start[k] = block[i][j].node[1],
Let end[k] = block[i][j].node[2]

Add (start[k], end[k]) to N’(S), Increment k by 1

Else
Table 5.5 (Continued)

If CYCNO % 2 = 0 Then //Check for even or odd cycle
    position = 2
Else
    position = 1
EndIf

p = position

While ( p< block[i][j].count)
    Let start[k] = block[i][j].node[p]
    Let end[k] = block[i][j].node[p+1]
    Add (start[k], end[k]) to N’(S), Increment k and p by 1
EndWhile
EndIf
EndFor
EndFor
EndIf
End
NWC strategy constructs a set of neighbors with a set of nodes in a block according to the current iteration number and a variation parameter v0. For odd cycle, odd neighbors and for even cycle, even neighbors are selected from a block. For example, if a block contains a set of nodes \{1, 2, 3, 4, 5, 6\}, a set of neighbors from this block has been found at \{(1, 2), (2, 3), (3, 4), (4, 5), (5, 6)\}. A set with odd neighbors is \{(1, 2), (3, 4), (5, 6)\}.

Elements in the set of odd neighbors are corresponding to locations of first, third and fifth in the original set. A set with even neighbors is \{(2, 3), (4, 5)\}. Elements in the set of even neighbors are corresponding to locations of second and fourth in the original set. Hence, sets of odd
and even neighbors have the elements corresponding to odd and even locations in the original set. Here the variation parameter \( v_0 \) is set from 1 to 10 and according to the value of the variation parameter \( v_0 \), all neighbors are selected in a particular iteration. That is, once in \( v_0 \) number of iterations, all nodes in a block are considered for the construction of a set of neighbors so that the original set of neighbors is produced for the current iteration.

In this section, computational results are given for well-known JSSP instances with initial tabu length as \( m+n \) and variation parameter value \( v_0 \) as 5. Table 5.6 shows a comparison of optimal values produced from different neighborhood
strategies for problem instances LA01-LA20 (Lawrence 1984), FT06 and FT10 (Fisher et al 1963), and ABZ5 and ABZ6 (Adams et al 1988). Column 1 specifies these problem instances. Column 2 shows the size of the problems. Column 3 shows the optimal value for each problem. Columns 4, 5, 6 and 7 specify results from FLN, MN, ALL and NWC neighborhood strategies respectively. It shows that tabu search using NWC strategy has succeeded in getting the optimal solutions for some problems and also in improving upper bound values compared with other methods.
<table>
<thead>
<tr>
<th>Problem Instance</th>
<th>Problem size</th>
<th>Opt</th>
<th>FLN strategy</th>
<th>MN strategy</th>
<th>ALL strategy</th>
<th>NWC strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td>LA01</td>
<td>10 × 5</td>
<td>666</td>
<td>666</td>
<td>666</td>
<td>666</td>
<td>666</td>
</tr>
<tr>
<td>LA02</td>
<td>10 × 5</td>
<td>655</td>
<td>655</td>
<td>655</td>
<td>655</td>
<td>655</td>
</tr>
<tr>
<td>LA03</td>
<td>10 × 5</td>
<td>597</td>
<td>585</td>
<td>630</td>
<td>610</td>
<td>597</td>
</tr>
<tr>
<td>LA04</td>
<td>10 × 5</td>
<td>590</td>
<td>595</td>
<td>635</td>
<td>615</td>
<td>590</td>
</tr>
<tr>
<td>LA05</td>
<td>10 × 5</td>
<td>593</td>
<td>593</td>
<td>593</td>
<td>593</td>
<td>593</td>
</tr>
<tr>
<td>LA06</td>
<td>15 × 5</td>
<td>926</td>
<td>925</td>
<td>943</td>
<td>918</td>
<td>926</td>
</tr>
<tr>
<td>LA07</td>
<td>15 × 5</td>
<td>890</td>
<td>945</td>
<td>937</td>
<td>925</td>
<td>890</td>
</tr>
<tr>
<td>LA08</td>
<td>15 × 5</td>
<td>863</td>
<td>886</td>
<td>980</td>
<td>945</td>
<td>870</td>
</tr>
<tr>
<td>LA09</td>
<td>15 × 5</td>
<td>951</td>
<td>945</td>
<td>945</td>
<td>945</td>
<td>938</td>
</tr>
<tr>
<td>LA10</td>
<td>15 × 5</td>
<td>958</td>
<td>965</td>
<td>965</td>
<td>965</td>
<td>958</td>
</tr>
<tr>
<td>LA11</td>
<td>20 × 5</td>
<td>1222</td>
<td>1228</td>
<td>1249</td>
<td>1227</td>
<td>1200</td>
</tr>
<tr>
<td>LA12</td>
<td>20 × 5</td>
<td>1039</td>
<td>1028</td>
<td>1042</td>
<td>1035</td>
<td>1039</td>
</tr>
<tr>
<td>LA13</td>
<td>20 × 5</td>
<td>1150</td>
<td>1137</td>
<td>1255</td>
<td>1148</td>
<td>1148</td>
</tr>
</tbody>
</table>
Table 5.5 (Continued)

<table>
<thead>
<tr>
<th>Problem Instance</th>
<th>Problem size</th>
<th>Opt</th>
<th>FLN strategy</th>
<th>MN strategy</th>
<th>ALL strategy</th>
<th>NWC strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td>LA14</td>
<td>20 × 5</td>
<td>1292</td>
<td>1308</td>
<td>1308</td>
<td>1308</td>
<td>1308</td>
</tr>
<tr>
<td>LA15</td>
<td>20 × 5</td>
<td>1207</td>
<td>1235</td>
<td>1289</td>
<td>1280</td>
<td>1217</td>
</tr>
<tr>
<td>LA16</td>
<td>10 × 10</td>
<td>945</td>
<td>945</td>
<td>945</td>
<td>945</td>
<td>945</td>
</tr>
<tr>
<td>LA17</td>
<td>10 × 10</td>
<td>784</td>
<td>790</td>
<td>996</td>
<td>795</td>
<td>778</td>
</tr>
<tr>
<td>LA18</td>
<td>10 × 10</td>
<td>848</td>
<td>860</td>
<td>950</td>
<td>878</td>
<td>855</td>
</tr>
<tr>
<td>LA19</td>
<td>10 × 10</td>
<td>842</td>
<td>842</td>
<td>860</td>
<td>842</td>
<td>842</td>
</tr>
<tr>
<td>LA20</td>
<td>10 × 10</td>
<td>902</td>
<td>934</td>
<td>984</td>
<td>941</td>
<td>920</td>
</tr>
<tr>
<td>FT06</td>
<td>6 × 6</td>
<td>55</td>
<td>53</td>
<td>53</td>
<td>53</td>
<td>53</td>
</tr>
<tr>
<td>FT10</td>
<td>10 × 10</td>
<td>930</td>
<td>930</td>
<td>930</td>
<td>930</td>
<td>930</td>
</tr>
<tr>
<td>ABZ5</td>
<td>10 × 10</td>
<td>1234</td>
<td>1270</td>
<td>1387</td>
<td>1284</td>
<td>1252</td>
</tr>
<tr>
<td>ABZ6</td>
<td>10 × 10</td>
<td>943</td>
<td>943</td>
<td>943</td>
<td>943</td>
<td>943</td>
</tr>
</tbody>
</table>
Figure 5.1 shows makespan values generated by the FLN, MN, ALL and NWC neighborhood strategies for different problem instances due to Lawrence (1984), Fisher et al (1963) and Adams et al (1988). MN strategy produces the worst results and NWC strategy produces the best results. Moderate results are generated by the FLN and ALL strategies and moreover FLN strategy is better than ALL strategy in terms of the quality of solution.
Figure 5.1 Makespan Values by Different Neighborhood Strategies for Different Problem Instances

5.9 DYNAMIC TABU LENGTH STRATEGY

Tabu length is changed during the solution construction phase to increase exploration of the search space. A new
dynamic tabu length strategy is proposed in this thesis, and this strategy finds the tabu length dynamically according to the iteration number. Procedure Dynamic Tabu With Cycles( ) is represented Table 5.7, an algorithm for this strategy and short form of this strategy is denoted by DTWC.

Table 5.7 Algorithm 6: Dynamic tabu with cycle()

<table>
<thead>
<tr>
<th>Algorithm: Dynamic Tabu With Cycles( )</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Inputs:</strong></td>
</tr>
<tr>
<td>CYCNO – Current iteration number</td>
</tr>
<tr>
<td>OLDL – Tabu length from previous iteration</td>
</tr>
<tr>
<td>range – Range of iterations within total iterations</td>
</tr>
<tr>
<td>d1, d2 – Control variables to change tabu length</td>
</tr>
<tr>
<td>r – Control variable to find the position of range within total iterations</td>
</tr>
<tr>
<td>u – Control variable to find the position of the current iteration within a range of iterations</td>
</tr>
</tbody>
</table>
### Table 5.7 (Continued)

**Algorithm: Dynamci Tabu With Cycles( )**

**Output:**
- TL – Tabu length for next iteration
- u – Updated control variable

**Begin**
- If CYCNO < range Then //Check for first range
  - Assign OLDL to TL
- Else
  - Assign false to set
- While (set = false) //Check for subsequent u ranges
  - If CYCNO >= (r*range) and CYCNO < (r*range + u*d1) Then
    - Compute TL = OLDL + u * d2 //Increment TL by d2 unit
    - Assign true to set
  - Else
    - Increment u by 1 //Go to next u\textsuperscript{th} range
- EndIf
- EndWhile
- EndIf
End
Current iteration number denoted by CYCNO, different control variables range, d1, d2, r and u are given as inputs. The control variables range; d1 and d2 are calculated as given in Equations (5.1), (5.2) and (5.3) respectively. Integral parts of these variables are used for the processing. The control variables r and u are used to find the position of the current iteration within the range interval.

\[
\text{range} = \frac{\text{MAXCYCNO}}{2\times m} \quad (5.1)
\]

\[
d1 = \frac{\text{range}}{m+n} \quad (5.2)
\]

\[
d2 = \frac{(d1+2\times m)}{m+n} \quad (5.3)
\]

The value of tabu length (TL) is m+n for the first range of iterations. For even and odd range intervals, TL value is
increased and decreased respectively by the value of d2 with a subsequent interval value of d1. This strategy improves the performance of TS during the construction of the solution. The change of the tabu length value is illustrated in Figure 5.2.

Figure 5.2 Illustration of Dynamic Tabu Length
DTWC strategy can be included in the algorithm TS_Procedure() described in section 5.2 by having following changes:

In step 1 of TS_Procedure(), the value of control variables r and u are initialized by 1 and the values of the range, d1 and d2 are determined. At the end of step 4, the following code can be added.

Assign TL to OLDL
If CYCNO % range = 0 Then
    Increment r by 1
Assign 1 to u
Assign -d2 = d2 //Negate d2
EndIf

Well-known JSSP instances are tested using the proposed NWC and DTWC
strategies with initial tabu length as m+n. Table 5.8 shows upper bound values obtained for problem instances LA01-LA20 (Lawrence 1984) and the relative error is calculated for the test instances. By using FTS approach with new neighborhood and dynamic tabu length strategies are most of the problems solved to the optimality, and other problems have been solved to near optimal values. Figure 5.3, 5.4 & 5.5 shows Proposed FTS approach using NWC and DTWC strategies.

Table 5.8 Performance of FTS Procedure using NWC and DTWC Strategies

<table>
<thead>
<tr>
<th>Problem instance</th>
<th>Problem size</th>
<th>Opt</th>
<th>UB by FTS with NWC and DTWC Strategies</th>
<th>Relative Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>LA01</td>
<td>10 × 5</td>
<td>666</td>
<td>666</td>
<td>0.00</td>
</tr>
</tbody>
</table>
Table 5.8 (Continued)

<table>
<thead>
<tr>
<th>Problem instance</th>
<th>Problem size</th>
<th>Opt</th>
<th>UB by FTS with NWC and DTWC Strategies</th>
<th>Relative Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>LA02</td>
<td>10 × 5</td>
<td>655</td>
<td>655</td>
<td>0.00</td>
</tr>
<tr>
<td>LA03</td>
<td>10 × 5</td>
<td>597</td>
<td>597</td>
<td>0.00</td>
</tr>
<tr>
<td>LA04</td>
<td>10 × 5</td>
<td>590</td>
<td>590</td>
<td>0.00</td>
</tr>
<tr>
<td>LA05</td>
<td>10 × 5</td>
<td>593</td>
<td>593</td>
<td>0.00</td>
</tr>
<tr>
<td>LA06</td>
<td>15 × 5</td>
<td>926</td>
<td>926</td>
<td>0.00</td>
</tr>
<tr>
<td>LA07</td>
<td>15 × 5</td>
<td>890</td>
<td>890</td>
<td>0.00</td>
</tr>
<tr>
<td>LA08</td>
<td>15 × 5</td>
<td>863</td>
<td>863</td>
<td>0.00</td>
</tr>
<tr>
<td>LA09</td>
<td>15 × 5</td>
<td>951</td>
<td>951</td>
<td>0.00</td>
</tr>
<tr>
<td>LA10</td>
<td>15 × 5</td>
<td>958</td>
<td>958</td>
<td>0.00</td>
</tr>
<tr>
<td>LA11</td>
<td>20 × 5</td>
<td>1222</td>
<td>1222</td>
<td>0.00</td>
</tr>
<tr>
<td>LA12</td>
<td>20 × 5</td>
<td>1039</td>
<td>1039</td>
<td>0.00</td>
</tr>
<tr>
<td>LA13</td>
<td>20 × 5</td>
<td>1150</td>
<td>1150</td>
<td>0.00</td>
</tr>
<tr>
<td>LA14</td>
<td>20 × 5</td>
<td>1292</td>
<td>1292</td>
<td>0.00</td>
</tr>
<tr>
<td>LA15</td>
<td>20 × 5</td>
<td>1207</td>
<td>1207</td>
<td>0.00</td>
</tr>
<tr>
<td>LA16</td>
<td>10 × 10</td>
<td>945</td>
<td>945</td>
<td>0.00</td>
</tr>
<tr>
<td>LA17</td>
<td>10 × 10</td>
<td>784</td>
<td>784</td>
<td>0.00</td>
</tr>
<tr>
<td>LA18</td>
<td>10 × 10</td>
<td>848</td>
<td>848</td>
<td>0.00</td>
</tr>
<tr>
<td>LA19</td>
<td>10 × 10</td>
<td>842</td>
<td>842</td>
<td>0.00</td>
</tr>
</tbody>
</table>
Figure 5.3 Comparisons between Makespan Vs Iterations of LA01

Figure 5.4 Comparisons between Makespan Vs Iterations of LA05
5.10 SUMMARY

The efficiency of the results obtained by the proposed TS for small, medium and large-sized instances is evaluated by considering the corresponding overall satisfactory level of all objectives.
Also the adaptability of the yielded solutions of the proposed TS for the small-sized instances is evaluated by comparing the results reported by the Lingo software. Several experiments on different sized test problems are considered and the related results are indicated the ability of the proposed TS algorithm to converge to the efficient solutions. The combination of fuzzy with TS meta-heuristic approach to solve JSSP. Improvement of the constructed schedule using the proposed method has been presented. Neighborhood solutions play a major role to generate good quality solutions. Different neighborhood structures have been evaluated and the best structure has been selected. New dynamic tabu length strategy has been proposed and different control variable has been employed to
change the length of Tabu search list. The algorithm has been tested using different benchmark problems. The performance of the algorithm has been compared with their pure parents and existing approaches. The algorithm produces the best results among others. Almost, the value of the over all satisfactory level of the TS solutions in medium to large-sized test problems was close to one (or equal to one). This implies the efficiency of the yielded solutions by the proposed TS method.