CHAPTER 6
CHAPTER 6

ITERATED LOCAL IMPROVEMENT EVOLUTIONARY ALGORITHMS

6.1 INTRODUCTION

According to the survey of Thomas Baeck et al. (1991), the Evolution Strategies community has always placed more emphasis on mutation than crossover [151]. The role of local search in the context of evolutionary algorithms and the wider field of evolutionary computing has been much discussed. In its most extreme form, this view casts mutation and other local operators as mere adjuncts to recombination, playing auxiliary (if important) roles such as keeping the gene pool well stocked and helping to tune final solutions. Radcliffe and Surry. (1994) investigated that a greater role for mutation, hill-climbing and local refinements are needed for evolutionary algorithms [152]. Ackley (1987) recommends genetic hill climbing, in which crossover plays a rather less dominant role [153]. This chapter addresses the development of a hybrid search algorithm named as iterated local improvement evolutionary algorithm (ILIEA) with population size (POP) equal to two.
Iterated local improvement evolutionary algorithm is designed similar to an iterated local improvement algorithm with evolutionary based perturbation tool. Iterated local improvement algorithm is a simple but effective procedure to explore multiple local minima, which can be implemented in any type of local search algorithm. It is to perform multiple runs with the algorithm and each using a different starting solution. A promising but relatively unexplored idea is to restart near a local optimum, rather than from a randomly generated solution. Under this approach, the next starting solution is obtained from the current local optimum where the current local optimum is usually either the best local optimum found so far from the history, or the most recently generated local optimum by applying a pre-specified type of random move to it which is referred as kick or perturbation. The algorithm skeleton of an iterated local improvement algorithm is given in figure 6.1.

```
Procedure Iterated Local Improvement
Begin
  $s_0$: initial base sequence;
  $i := 0$;
  while (¬termination criteria ($i$)) do
    begin
      $s^* := \text{Local Improvement} (s_0)$;
      $s' := \text{Perturbation} (s^*, OX, \text{history})$
      $s^{**} := \text{Local Improvement} (s')$
      $s^* := \text{acceptance criteria} (s^*, s^{**}, \text{history})$
    end
  end
```

Figure 6.1 Iterated Local Improvement Algorithms
The iterated local improvement algorithm requires the following components:

- Generation of initial sequence
- Local Improvement Technique
- Perturbation
- Acceptance Criterion
- Termination Criterion

Iterated local improvement Evolutionary algorithms are developed based on the iterated local improvement algorithm and have the following new features:

- Evolutionary perturbation tool in the crossover operation
- Self Improvement in Mutation operation

In this chapter, the development of a hybrid iterated local improvement evolutionary algorithm to solve the benchmark instances single machine total weighted tardiness problems is discussed.

6.2 LOGIC OF PROPOSED ALGORITHM

The logic behind the development of iterated local improvement evolutionary algorithm involves as follows:

- Application of sequence developed by backward heuristic
- Application of a sequence generated randomly
- Application of evolutionary perturbation technique in crossover operation
- Application of self improvement technique in mutation operation
- Application of a relation between the sizes of the problem in termination criterion

*Application of sequence developed by backward heuristic*

The heuristic improvement algorithm described in the previous chapter, takes the sequence generated by backward heuristic as initial sequence, is not performing well for
higher sizes of problems and shows improvement for smaller size problems. So, it is proposed to adopt the sequence generate by this dynamic dispatching rule as one of the initial parents for the proposed iterated local improvement evolutionary algorithm.

*Application of a sequence generated randomly*

Random initialization of population is adopted by many researchers [145, 154] in evolutionary approaches and it is proposed to adopt a sequence generated by taking events in a random order as other initial parent.

*Application of evolutionary perturbation technique in crossover operation*

It is proposed to apply the concept of iterated local search with evolutionary perturbation tool in the crossover operation of the proposed algorithm. It builds a new sequence by a pre-specified move called as perturbation from the initial sequence. An ordered crossover operator (OX) is used as the evolutionary perturbation tool. The number of components modified in the sequence is defined as perturbation strength is more in an ordered cross over compared to other perturbation tools. The evolutionary perturbation tool is operating on the two initial parents and a new offspring or sequence is build.

*Application of self improvement technique in mutation operation*

The perturbated sequence obtained from the crossover operation is improved by a local search which is a trajectory method. Local search methods involve repeatedly going from one solution to another by a *local-move*. The set of all solutions reachable from a solution $s$ through a *local-move* is termed as the *neighborhood* of $s$ $N(s)$. Pilot experiments were conducted in local search with different mechanisms to generate neighbors and it is proposed to adopt best mechanism in the mutation for improvement. It is observed that the randomized pair wise interchange mechanism is performing well and it is used as mutation operator in the proposed algorithm.
Application of a relation between the sizes of the problem in termination criterion

In most of the evolutionary algorithms, the termination criteria may assume to be number of generations and in this algorithm number of generations is assumed to be a function of the problem size. The complexity of the algorithm can be identified easily by this assumption.

6.3 ITERATED LOCAL IMPROVEMENT EVOLUTIONARY ALGORITHM

Iterated Local Improvement Evolutionary Algorithm (ILIEA) is hybrid algorithm having $POP = 2$. The complexity of the algorithm is governed by the number of iterations used for termination criterion. The complete process of iterated local improvement evolutionary algorithm with an example is given in the figure 6.2. It consists of the following modules:

- Initial parents generation
  
  Population size $POP = 2$

- Crossover operation (Evolutionary perturbation technique)
  
  Crossover probability $(P_c) = 1$

- Mutation operation (Self improvement technique)
  
  Mutation probability $(P_m) = 1$

- New parents generation
6.3.1 Initial Parents Generation

A sequence of the single machine scheduling problem is mapped into a chromosome with the alleles assuming different and non repeating integer values in the \([1,n]\) interval. Any sequence can be mapped into this permutation representation. This approach can be found in most genetic algorithm articles dealing with sequencing problems [145]. The total weighted tardiness of a sequence is assumed to be the fitness function for \textit{ILIEA}.

In this algorithm the population size is assumed to be two and the sequence developed by the backward phase acts as one parent and sequence generated taking events in a random order acts as the other parent.
6.3.2 Crossover Operation (Evolutionary Perturbation Technique)

The crossover operation adopted in this thesis uses an evolutionary perturbation technique, which is explained as follows:

Perturbation is a pre-specified type of random move is applied to a solution. For a current solution $s^*$, a change or perturbation is applied to an intermediate state $s'$. Then the *Local Improvement* is applied on $s'$ and a new solution $s^{**}$ is reached. If $s^{**}$ passes an acceptance test, it becomes the next base solution for the search otherwise it returns to $s^*$. The overall procedure is shown in figure 6.3.

![Figure 6.3 Procedures for Perturbation](image)

**Figure 6.3 Procedures for Perturbation**

The process of perturbation is having the following parts:

- Iterated local search (*ILS*)
- Perturbation tool
- Perturbation strength
- Acceptance criterion
6.3.2.1 Iterated Local Search

The underlying idea of ILS is that of building a random walk in $S^*$, the space of local optima defined by the output of a given local search. Four basic ingredients are needed to derive an ILS:

- a procedure $\text{GenerateInitialSolution}$, which returns some initial solution,
- a local search procedure $\text{LocalSearch}$,
- a scheme of how to perturb a solution, implemented by a procedure $\text{Perturbation}$, and
- an $\text{AcceptanceCriterion}$, which decides from which solution the search is continued.

The particular walk in $S^*$ followed by the ILS can also depend on the search history, which is indicated by $\text{history}$ in $\text{Perturbation}$ and $\text{AcceptanceCriterion}$.

The effectiveness of the walk in $S^*$ depend on the definition of the four component procedures of ILS: The effectiveness of the local search is of major importance, because it strongly influences the final solution quality of ILS and its overall computation time. The perturbations should allow the ILS to effectively escape local optima but at the same time avoid the disadvantages of random restart. The acceptance criterion, together with the perturbation, strongly influence the type of walk in $S^*$ and can be used to control the balance between intensification and diversification of the search. The initial solution will be important in the initial part of the search. The configuration problem in ILS is to find a best possible choice for the four components such that best overall performance is achieved. The algorithm outline of iterated local search is given in the figure 6.4.
Outline of Iterated Local Search

\[ s_0 = \text{GenerateInitialSolution} \]
\[ s^* = \text{LocalSearch}(s_0) \]

repeat
\[ s' = \text{Perturbation}(s^*, \text{history}) \]
\[ s'' = \text{LocalSearch}(s') \]
\[ s^* = \text{AcceptanceCriterion}(s^*, s'', \text{history}) \]
until
termination criterion met

Figure 6.4 Iterated Local Search

6.3.2.2 Perturbation Tool

Though many researchers followed different types of perturbation tools, an evolutionary operator perturbation tool is used in this work. Here, an ordered crossover operator (OX) is used as perturbation tool. The operation of the OX is given as follows: The operator takes the initial sequence \( s^* \) from the base heuristics and another sequence \( s^{**} \) is generated randomly. The resultant sequence \( s' \) will take, a fragment of the sequence from \( s^* \) and the selection of the fragment is made uniformly at random. In the second phase, the empty positions of \( s' \) are sequentially filled according \( s^{**} \). The accepted \( s^* \) for the next iteration will replace with worst of the previous \( s^* \) and \( s^{**} \).

As an example, the sequence \( s' \) inherits the elements between the two crossover points, inclusive, from \( s^* \) in the same order and position as they appeared. The length of the crossover is in the range between a random number generated in the range of \([1, n-1]\) job position as lower limit (LL) and a random number generated in the range of \([LL, n]\) as the upper limit (UL). The remaining elements are inherited from the alternate sequence \( s^{**} \) in
the order in which they appear, beginning with the first position following the second crossover point and skipping over all elements already present in \( s' \).

An example for the perturbation tool is given in figure 6.5. The elements \( \alpha, \Gamma, \nu, \delta \) and \( \omega \) are inherited from \( s^* \) in the same order and position in which they occur. Then, starting from the first position after the second crossover point, \( s' \) inherits from \( s^{**} \). In this example, position 8 the next position, \( s'[8] = \nu \), which is already present in the offspring, so \( s^{**} \) is searched until an element is found which is not already present in \( s' \). Since \( \nu, \omega \) and \( \Gamma \) are already present in \( s' \), the search continues from the beginning of the string and \( s'[8] = s^{**}[2] = \beta, s'[9] = s^{**}[3] = \gamma, s'[10] = s^{**}[5] = \epsilon \), and so on until the new sequence is generated [155].

![Parent 1 (s*)](image)

Parent 1 (s*): \( \gamma - \zeta - \alpha - \Gamma - \nu - \delta - \omega - \lambda - \beta - \epsilon \)

Parent 2 (s**): \( \alpha - \beta - \gamma - \delta - \epsilon - \zeta - \lambda - \nu - \omega - \Gamma \)


Offspring (s'): \( \zeta - \lambda - \alpha - \Gamma - \nu - \delta - \omega - \beta - \gamma - \epsilon \)

**Figure 6.5 Ordered Crossover (OX)**

6.3.2.3 Perturbation Strength

For some problems, appropriate perturbation strength is very small and seems to be rather independent of the instance size. The strength of a perturbation is referred as the number of solution components directly affected by a perturbation. The OX operator will change most of the solution components in the sequence according to the generated \( LL \) & \( UL \) values.
6.3.2.4 Acceptance Criteria

The perturbation mechanism together with the local improvement defines the possible transitions between a current solution \( s^* \) to a "neighboring" solution \( s^* \). The acceptance criteria determines whether \( s^* \) is accepted or not as the new current solution. A natural choice for the acceptance criterion is to accept only better solutions which are a very strong intensification for search. This is termed as BETTER criterion. Diversification of the search is extremely favored if every \( s^* \) is accepted as the new solution. This is termed as random walk (\( RW \)) criterion which is represented as

\[
RW(s^*, s^*, \text{history}) : = s^* \]  \hspace{1cm} (6.1)

Since, the operator \( OX \) completely changes most of the solution components, the acceptance criterion is chosen as \( RW \).

The sequence obtained after perturbation is further improved in the mutation operation which is self improving.

6.3.3 Mutation Operation (Self Improvement Technique)

The mutation operation adopted in this thesis uses a self improvement technique, which is explained as follows:

This self improvement technique is the method to improve the solution for predetermined trials of iterations. This technique is developed from local search method which is a trajectory method and searches the solution space non-systematically until a specific stop criterion is satisfied. The self improvement technique consists of the following parts:
• Local search
• Neighborhood structure

6.3.3.1 Local Search

Local search methods move iteratively through the solution set \( S \). Based on the current and may be on the previous visited solutions, a new solution is chosen. The choice of the new solution is restricted to solutions that are somehow close to the current solution i.e. in the 'neighborhood' of the current solution. Different local search methods may be formulated depending on the method of choosing solutions from the neighborhood of the current solution and the way in which the stopping criteria are defined [156]. A neighborhood search method requires a representation of solutions to be chosen, and an initial solution to be constructed by some heuristic rule or created randomly. A neighbor is generated by some suitable mechanism, and an acceptance rule is used to decide whether it should replace the current solution or not. The acceptance rule in a neighborhood search method usually requires the comparison of objective function values for the current solution and its neighbor.

Neighborhoods are usually defined by first choosing a simple type of transition to obtain a new solution from a given one, and then defining the neighborhood as the set of all solutions that can be obtained from a given solution by performing one transition. Generally, a local search method is based on the following two routines:

- Given an instance, construct an initial solution.
- Given an instance and any solution, determine if there is a neighboring solution of lower cost, and if so, return one such solution. If no such solution exists, then the input solution is returned and it is indicated that it is a local optimal solution.

The basic structure of a local search is presented in figure 6.6
Procedure Local Search (Search Space \(S\), Neighborhood \(N\), \(Z(\sigma)\));

begin
\[ \sigma_0 : = \text{Initial sequence (}\sigma\text{);} \]
\[ i : = 0; \]

while (termination criteria (\(\sigma_i\ i\))) do

\[ m : = \text{Selectmove (}\sigma_i N, Z(\sigma_i)\text{);} \]
if \(Z'(\sigma) > Z(\sigma)\)
then \(\sigma_{i+1} = \sigma_i \circ m;\)
else \(\sigma_{i+1} = \sigma_i;\)
\[ i = i + 1 \]

end
end;

Figure 6.6 Local Search

6.3.3.2 Neighborhood Structure

Before applying local search methods to any problem a neighborhood structure is to be defined. A systematic way of defining neighborhoods is needed; otherwise, it is not possible to store the neighborhood. The neighborhoods define a frame for the possibilities of walking through the solution space; they have a crucial influence on the behavior of local search. If neighborhoods are small, the walk is very restricted and, thus, it may be hard to reach good solutions. On the other hand, if neighborhoods are large, it may be time consuming to decide in which direction (i.e. to which neighbor) the search shall continue. However, not only the size but the more the quality of the solutions in a neighborhood is of interest. If a neighborhood contains promising solutions, it does not matter if the size of the
neighborhood is small and, on the other hand, large neighborhoods with only solutions of poor quality are not very helpful.

Three common neighborhood schemes are used for scheduling problems and are given below:

- **Adjacent neighborhood interchange** in which a job may be swapped with jobs directly to its left or right in the schedule.
- **Swap** in which any two jobs in the schedule can be swapped.
- **Insert** in which a job is taken from its current position and placed in another position in the schedule.

In this thesis, four mechanisms are used for finding the neighborhood solutions to solve the single machine scheduling problems are investigated. They are:

- Adjacent neighborhood interchange
- Randomized neighborhood structure
  - Randomized adjacent interchange ($\psi_{ai}$),
  - Randomized sliding mutation ($\psi_{sl}$) and
  - Randomized pair wise interchange ($\psi_{pw}$)

*Adjacent neighborhood interchange*

The process of the *adjacent neighborhood interchange* mechanism is shown in figure 6.7. For any solution $s$, neighbourhood of $s$, $N(s)$, includes $(n-1)$ different alternative neighbouring solutions obtained by interchanging a job with its right job in the sequence.
Randomized Adjacent Interchange ($\psi_{ai}$)

This is a randomized version of adjacent interchange neighborhood structure. This operator will generate a random number ($R$) in the range $[1, n]$ and just interchanges the job present in the position $R$ with the next job in the sequence ($R+1$) and represented as:

$$\psi_{ai}(\pi_j^{\omega}) = \pi_{j+1}$$

--- (6.2)

Randomized Sliding Mutation ($\psi_{si}$)

This is a randomized version of inert neighborhood structure. This operator may be also termed as randomized extraction and backward shift insertion operator. Sliding mutation refers to "moving a job from the $j^{th}$ place and placing it before the $i^{th}$ position". Two values are generated randomly ($R_1$ and $R_2$) in the range $[1,n]$ in such a way that $R_1 < R_2$ and applied to jobs present in the positions in between $R_1$ and $R_2$. The job in position $R_2$ is
placed before the job in position $R_1$ and all jobs in between $R_1$ and $R_2$ are pushed one position and represented as:

$$\psi_{nl}(\omega) = \nu j i \pi \omega \quad \text{--- (6.3)}$$

*Randomized Pair wise Interchange ($\psi_{pw}$)*

This operator may be also termed as random swap operator and similar to swap neighborhood structure. Random swap refers to "the swapping according to the randomly generated values". Two values are generated randomly ($R_1$ and $R_2$) in the range $[1,n]$ and applied to jobs present in the positions $R_1$ and $R_2$ and the jobs are swapped according to the random values generated and represented as:

$$\psi_{pw}(\omega) = \nu j i \pi \omega \quad \text{--- (6.4)}$$

The improvement technique will be stopped with a maximum number of trials which is assumed to be a function related to number of jobs ($n$).

*Performance Comparison local search with different Neighborhood Structures*

The local search with different neighborhood structures is coded in C++ on a personal computer with 1.3 GHz Pentium IV CPU and 128 MB main memory running on Microsoft Windows operating system 2000 (5 RELEASE version) with Borland C/C++ compiler (version 3.1).
The local search with different neighborhood structures with a termination criteria \( n^*n^*n \) number of iterations, so that the complexity of the algorithm is in the order of \( O(n^3) \), applied on the initial sequence obtained by backward phase heuristics.

The algorithm is tested on 125 benchmark instances of single machine total weighted tardiness problems explained in chapter 4 of each sizes \( n = 40, n = 50 \) & \( n = 100 \).

The \((RDD, TF)\) factor wise comparison of the results obtained by the local search with apparent interchange structure are given in Appendix.

It is observed that the local search algorithm with apparent neighborhood interchange is applied on the sequence generated by backward heuristics is not able to improve further and it is decided to use the randomized neighborhood structure.

The potentials of three randomized neighborhood structure are investigated by applying on the sequences generated by the \( EDD, MDD \) and \( BH \) heuristics as initial sequences. These local search is applied for a termination criteria \( n^*n^*n \) number of iterations so that the complexity of the algorithm is in the order of \( O(n^3) \). The effect on the average total weighted tardiness of the 125 problem instances of each sizes on \( n = 40, n = 50 \) and \( n = 100 \) are compared and given in the table 6.1.

**Table 6.1 Average total weighted tardiness comparison of transition move operators**

<table>
<thead>
<tr>
<th>( n )</th>
<th>EDD</th>
<th>MDD</th>
<th>BH</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \psi_{pw} )</td>
<td>( \psi_{st} )</td>
<td>( \psi_{si} )</td>
<td>( \psi_{pw} )</td>
</tr>
<tr>
<td>40</td>
<td>425756.3</td>
<td>438103.7</td>
<td>450255.5</td>
</tr>
<tr>
<td>50</td>
<td>94622.33</td>
<td>102627.5</td>
<td>95200.38</td>
</tr>
<tr>
<td>100</td>
<td>425756.3</td>
<td>438103.7</td>
<td>450255.5</td>
</tr>
</tbody>
</table>
It is inferred from the table 6.1, that the local search with the randomized pair wise interchange ($\psi_{pw}$) structure is performing better than the other two. Further, it is observed that improvement on EDD initial sequence is giving good results for size $n = 40$ and for sizes $n = 50$ and $n = 100$ improvement on BH initial sequence is performing well. So, it indicates for large sizes of $n$, $\psi_{pw}$ structure can be applied as self improving technique in this proposed iterated local improvement evolutionary algorithm with a maximum number of trials for local improvement, which can be assumed as a function of size of the problem.

6.3.4 New Parent Generation

In this proposed algorithm, the locally improved offspring obtained after self improvement technique is used as a parent for the next generation. Even though, the improved offspring value is less than the previous parents, it must be considered for the next generation. The best parent of the previous generation will act as the other parent and the evolution process is continued for the predetermined number of generation.

6.4 PERFORMANCE ANALYSIS OF ILIEA WITH AND WITHOUT CROSSOVER OPERATION

In the proposed iterated local improvement evolutionary algorithm, an evolutionary perturbation technique is used in the crossover operation and a simple experiment is conducted in ILIEA for testing the algorithm with and without crossover. The benchmark instances of single machine total weighted tardiness problems are solved by ILIEA with crossover and without crossover for different termination criteria. The termination criteria are assumed to be different number of iterations like 30,000, 50,000, 60,000, 80,000 and 1, 00,000. The average total weighted tardiness obtained by these algorithms for different sizes of the problems is given in table 6.2.
Table 6.2 Average total weighted tardiness - ILIEA without and with crossover

<table>
<thead>
<tr>
<th>S.No</th>
<th>No. of generations</th>
<th>n = 40</th>
<th>n = 50</th>
<th>n = 100</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Without perturbation</td>
<td>With perturbation</td>
<td>Without perturbation</td>
<td>With perturbation</td>
</tr>
<tr>
<td>1</td>
<td>30,000</td>
<td>49147.75</td>
<td>39740.88</td>
<td>69056.30</td>
</tr>
<tr>
<td>2</td>
<td>50,000</td>
<td>48972.22</td>
<td>38362.7</td>
<td>68827.24</td>
</tr>
<tr>
<td>3</td>
<td>60,000</td>
<td>48587.52</td>
<td>37967.91</td>
<td>68499.96</td>
</tr>
<tr>
<td>4</td>
<td>80,000</td>
<td>48353.4</td>
<td>37812.6</td>
<td>68127.16</td>
</tr>
<tr>
<td>5</td>
<td>100,000</td>
<td>48010.12</td>
<td>37745.35</td>
<td>67410.20</td>
</tr>
</tbody>
</table>

The average percentage of error is calculated by the formula

\[
\% \text{ of deviation} = \frac{Z_{ILIEA} - Z_{best}}{Z_{best}} \times 100\% \quad ---- \ (6.5)
\]

The average percentage of deviation from the best known solutions for the different number of generations is given in figure 6.8 for size \( n = 40 \), figure 6.9 for size \( n = 50 \) and figure 6.10 for \( n = 100 \).
Figure 6.8 Percentage of Deviation vs No. of Generations for $n = 40$
Figure 6.9 Percentage of Deviation vs No. of Generations for \( n = 50 \)
Figure 6.10 Percentage of Deviation vs No. of Generations for $n = 100$
From the figures, it is observed that the percentage of deviation from the best known values is decreasing for ILIEA if the evolutionary perturbation tool is used in the crossover operation.

6.5 PERFORMANCE ANALYSIS OF ILIEA

The iterated local improvement algorithm is coded in C++ on a personal computer with 1.3 GHz Pentium IV CPU and 128 MB main memory and running on Microsoft Windows operating system 2000 (5 RELEASE version) with Borland C/C++ compiler (version 3.1). They are tested on 125 benchmark instances of single machine total weighted tardiness problems of each sizes $n = 40$, $n = 50$ and $n = 100$.

In the single machine total weighted tardiness problems, there are 25 different combinations for $(RDD, TF)$ pairs and five replicates are taken for each $(RDD, TF)$ combinations yielding 125 different test instances for each value of $n$.

The average total weighted tardiness values of five replicates of each $(RDD, TF)$ combinations for the size $n = 40$ are considered and compared with the best known values available in the file $wtopt40$.

The $(RDD, TF)$ factor wise comparison of results of iterated local improvement evolutionary algorithm for size $n = 40$ is given in the table 6.3.
Table 6.3 (RDD, TF) factor wise comparison for \( n = 40 \) – ILIEA

<table>
<thead>
<tr>
<th>S.No</th>
<th>RDD</th>
<th>T.F.</th>
<th>Average weighted tardiness</th>
<th>% of deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Best known value</td>
<td>ILIEA</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1151.8</td>
<td>1190.6</td>
</tr>
<tr>
<td>1.</td>
<td>0.2</td>
<td>0.2</td>
<td>9221.2</td>
<td>9221.2</td>
</tr>
<tr>
<td>2.</td>
<td>0.2</td>
<td>0.4</td>
<td>21464.8</td>
<td>21464.8</td>
</tr>
<tr>
<td>3.</td>
<td>0.2</td>
<td>0.6</td>
<td>73120.2</td>
<td>73120.2</td>
</tr>
<tr>
<td>4.</td>
<td>0.2</td>
<td>0.8</td>
<td>112514</td>
<td>112514</td>
</tr>
<tr>
<td>5.</td>
<td>0.2</td>
<td>1.0</td>
<td>66.4</td>
<td>66.4</td>
</tr>
<tr>
<td>6.</td>
<td>0.4</td>
<td>0.2</td>
<td>4815.8</td>
<td>4833.2</td>
</tr>
<tr>
<td>7.</td>
<td>0.4</td>
<td>0.4</td>
<td>20039.8</td>
<td>20070</td>
</tr>
<tr>
<td>8.</td>
<td>0.4</td>
<td>0.6</td>
<td>69790.8</td>
<td>69999</td>
</tr>
<tr>
<td>9.</td>
<td>0.4</td>
<td>0.8</td>
<td>91736.8</td>
<td>91887.2</td>
</tr>
<tr>
<td>10.</td>
<td>0.6</td>
<td>0.2</td>
<td>3273.6</td>
<td>3303.4</td>
</tr>
<tr>
<td>11.</td>
<td>0.6</td>
<td>0.4</td>
<td>18541.2</td>
<td>18583</td>
</tr>
<tr>
<td>12.</td>
<td>0.6</td>
<td>0.6</td>
<td>71892.4</td>
<td>72006.8</td>
</tr>
<tr>
<td>13.</td>
<td>0.6</td>
<td>1.0</td>
<td>90276</td>
<td>90796.6</td>
</tr>
<tr>
<td>14.</td>
<td>0.8</td>
<td>0.2</td>
<td>609.4</td>
<td>633.8</td>
</tr>
<tr>
<td>15.</td>
<td>0.8</td>
<td>0.4</td>
<td>14593.8</td>
<td>14672</td>
</tr>
<tr>
<td>16.</td>
<td>0.8</td>
<td>0.8</td>
<td>49719.8</td>
<td>50817.2</td>
</tr>
<tr>
<td>17.</td>
<td>0.8</td>
<td>1.0</td>
<td>121667.6</td>
<td>121667.6</td>
</tr>
<tr>
<td>18.</td>
<td>1.0</td>
<td>0.2</td>
<td>774</td>
<td>780.4</td>
</tr>
<tr>
<td>19.</td>
<td>1.0</td>
<td>0.4</td>
<td>22629.2</td>
<td>22839.6</td>
</tr>
<tr>
<td>20.</td>
<td>1.0</td>
<td>0.6</td>
<td>51664</td>
<td>51664</td>
</tr>
<tr>
<td>21.</td>
<td>1.0</td>
<td>0.8</td>
<td>91482.4</td>
<td>91502.8</td>
</tr>
</tbody>
</table>

From the table 6.3, it is observed that the average percentage of deviation is 0.395% from the best known values for size \( n = 40 \).
The average total weighted tardiness values of five replicates of each \((RDD, TF)\) combinations for the size \(n = 50\) are considered and compared with the best known values available in the file \(wtopt50\).

Similarly, the \((RDD, TF)\) factor wise comparison of results of iterated local improvement evolutionary algorithm for size \(n = 50\) is given in the table 6.4.

**Table 6.4 \((RDD, TF)\) factor wise comparison for \(n = 50\) – ILIEA**

<table>
<thead>
<tr>
<th>S.No</th>
<th>RDD</th>
<th>T.F.</th>
<th>Average weighted tardiness</th>
<th>% of deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Best known value</td>
<td>ILIEA</td>
</tr>
<tr>
<td>1.</td>
<td>0.2</td>
<td>0.2</td>
<td>2184.4</td>
<td>2214.2</td>
</tr>
<tr>
<td>2.</td>
<td>0.2</td>
<td>0.4</td>
<td>13343.4</td>
<td>13523.2</td>
</tr>
<tr>
<td>3.</td>
<td>0.2</td>
<td>0.6</td>
<td>43196.8</td>
<td>43216.8</td>
</tr>
<tr>
<td>4.</td>
<td>0.2</td>
<td>0.8</td>
<td>87714.4</td>
<td>87749.4</td>
</tr>
<tr>
<td>5.</td>
<td>0.2</td>
<td>1.0</td>
<td>189113</td>
<td>189950.8</td>
</tr>
<tr>
<td>6.</td>
<td>0.4</td>
<td>0.2</td>
<td>176.4</td>
<td>176.4</td>
</tr>
<tr>
<td>7.</td>
<td>0.4</td>
<td>0.4</td>
<td>6452.4</td>
<td>7102.2</td>
</tr>
<tr>
<td>8.</td>
<td>0.4</td>
<td>0.6</td>
<td>32574.6</td>
<td>32588.6</td>
</tr>
<tr>
<td>9.</td>
<td>0.4</td>
<td>0.8</td>
<td>89835.2</td>
<td>90302.8</td>
</tr>
<tr>
<td>10.</td>
<td>0.4</td>
<td>1.0</td>
<td>166049.6</td>
<td>166274</td>
</tr>
<tr>
<td>11.</td>
<td>0.6</td>
<td>0.2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>12.</td>
<td>0.6</td>
<td>0.4</td>
<td>3426.6</td>
<td>3604.6</td>
</tr>
<tr>
<td>13.</td>
<td>0.6</td>
<td>0.6</td>
<td>23277.6</td>
<td>24065.2</td>
</tr>
<tr>
<td>14.</td>
<td>0.6</td>
<td>0.8</td>
<td>81545.4</td>
<td>81756.4</td>
</tr>
<tr>
<td>15.</td>
<td>0.6</td>
<td>1.0</td>
<td>130365</td>
<td>130731</td>
</tr>
<tr>
<td>16.</td>
<td>0.8</td>
<td>0.2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>17.</td>
<td>0.8</td>
<td>0.4</td>
<td>2191.2</td>
<td>2291.8</td>
</tr>
<tr>
<td>18.</td>
<td>0.8</td>
<td>0.6</td>
<td>25873.8</td>
<td>26188.8</td>
</tr>
<tr>
<td>19.</td>
<td>0.8</td>
<td>0.8</td>
<td>63134.6</td>
<td>63179.8</td>
</tr>
<tr>
<td>20.</td>
<td>0.8</td>
<td>1.0</td>
<td>153155.6</td>
<td>153227.6</td>
</tr>
<tr>
<td>21.</td>
<td>1.0</td>
<td>0.2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>22.</td>
<td>1.0</td>
<td>0.4</td>
<td>1839.4</td>
<td>1839.4</td>
</tr>
<tr>
<td>23.</td>
<td>1.0</td>
<td>0.6</td>
<td>20864.8</td>
<td>21067.6</td>
</tr>
<tr>
<td>24.</td>
<td>1.0</td>
<td>0.8</td>
<td>76158</td>
<td>76166.2</td>
</tr>
<tr>
<td>25.</td>
<td>1.0</td>
<td>1.0</td>
<td>109855.4</td>
<td>109908.6</td>
</tr>
</tbody>
</table>
From the table 6.4, it is observed that the average percentage of deviation is 0.748% from the best known values for size \( n = 50 \).

The average total weighted tardiness values of five replicates of each \((RDD, TF)\) combinations for the size \( n = 100 \) are considered and compared with the best known values available in the file \textit{wtbest100a}. Similarly, the \((RDD, TF)\) factor wise comparison of results of iterated local improvement evolutionary algorithm for size \( n = 100 \) is given in the table 6.5.

\begin{table}[h]
\centering
\caption{Table 6.5 \((RDD, TF)\) factor wise comparison for \( n = 100 \) – \textit{ILIEA}}
\begin{tabular}{|c|c|c|c|c|c|c|}
\hline
S.No & RDD & T.F. & \multicolumn{2}{|c|}{Average weighted tardiness} & \% of deviation \\
\cline{4-5}
\hline
\hline
1. & 0.2 & 0.2 & 5343.8 & 6180.4 & 15.656 \\
2. & 0.2 & 0.4 & 52570 & 53164.6 & 1.131 \\
3. & 0.2 & 0.6 & 185027.8 & 185835.2 & 0.004 \\
4. & 0.2 & 0.8 & 433824.6 & 436382.6 & 0.006 \\
5. & 0.2 & 0.1 & 665021.4 & 666331.8 & 0.002 \\
6. & 0.4 & 0.2 & 256.6 & 256.6 & 0.000 \\
7. & 0.4 & 0.4 & 24792.8 & 27262.8 & 9.963 \\
8. & 0.4 & 0.6 & 132402.4 & 137293.2 & 3.694 \\
9. & 0.4 & 0.8 & 374993.8 & 379095.6 & 1.093 \\
10. & 0.4 & 1.0 & 691626.8 & 703858.2 & 1.768 \\
11. & 0.6 & 0.2 & 0 & 0 & 0.000 \\
12. & 0.6 & 0.4 & 12955 & 14756 & 13.903 \\
13. & 0.6 & 0.6 & 85544.2 & 91407.6 & 6.854 \\
14. & 0.6 & 0.8 & 315179.2 & 330526.8 & 4.869 \\
15. & 0.6 & 1.0 & 607101.8 & 61426.4 & 0.007 \\
16. & 0.8 & 0.2 & 0 & 0 & 0.000 \\
17. & 0.8 & 0.4 & 656.6 & 695.4 & 5.909 \\
18. & 0.8 & 0.6 & 67259.2 & 71899.8 & 6.900 \\
19. & 0.8 & 0.8 & 295368.4 & 297195.6 & 0.006 \\
20. & 0.8 & 1.0 & 576902 & 578342.4 & 0.002 \\
21. & 1.0 & 0.2 & 0 & 0 & 0.000 \\
22. & 1.0 & 0.4 & 285 & 338.4 & 18.736 \\
23. & 1.0 & 0.6 & 132623 & 141838.2 & 6.948 \\
24. & 1.0 & 0.8 & 300435 & 303187.6 & 0.009 \\
25. & 1.0 & 1.0 & 486114.2 & 487220.8 & 0.002 \\
\hline
\end{tabular}
\end{table}
From the table, it is observed that the average percentage of deviation is 3.898% from the best known values for size \( n = 100 \).

6.6 SUMMARY

In this chapter, an iterated local improvement evolutionary algorithm to solve single machine scheduling problems to minimize the total weighted tardiness is presented. The algorithm uses only a single pair of parents; one sequence obtained from a greedy backward phase heuristic and the other by random generation act as the initial parents. The performance of this algorithm with and without crossover operation is compared. The problem instances available in the OR library and the best-known solution are used as the criteria for finding out the percentage of deviation. The average percentage of deviation is ranging from 27.54% to 38.28% for the iterated local improvement algorithm without perturbation and the ranging from 0.28% to 16.84% for the iterated local improvement evolutionary algorithm with perturbation.

The population of iterated local improvement evolutionary is increased from two and a self improving mutation evolutionary algorithm is formulated, is dealt in the next chapter.