Solving Composite Multi objective Single Machine Scheduling Problem Using Branch and Bound and Local Search Algorithms

Tariq S. Abdul – Razaq1, Hafed M. Motair2
1Department of Mathematics, College of Science, Mustansiriyah University, IRAQ
2Department of Mathematics, Open Educational College, Al- Qadisiya, Ministry of Education, IRAQ
*Correspondent Author Email: hafedmotair@gmail.com

Abstract
This paper present algorithm for solving a single machine scheduling problem to minimize the sum of total completion times, total tardiness, maximum tardiness, and maximum earliness. The single machine total tardiness problem is already NP-hard, so they consider problem is strongly NP-hard, and several algorithms are used to solve it. Branch and bound algorithm with dominance rule and local search algorithms are proposed for the problem. For the Branch and bound algorithm results- show that using dominance rule improve the performance of the algorithm in both computation times and optimal values, but it needs longer times. Thus we tackle the problem of large sizes with local search algorithms descent method, simulated annealing and tabu search. The performance of these algorithms is evaluated on a large set of test problems and the results are compared. The computational results show that simulated annealing algorithm and Tabu search algorithm are better than descent method with preference to simulated annealing algorithm, and show that the three algorithms find optimal or near optimal solutions in reasonable times.

Keywords: Multicriteria Scheduling, Branch and bound, Dominance rule, Local search algorithms.

Introduction
Scheduling concerns the allocation of limit resources to tasks over time. It is a decision-making process that has as a goal the optimization of one or more objectives [11]. Scheduling problems in real life applications generally involves optimization of more than one criteria. These criteria are often conflicting in nature and quite complex, according to how these criteria optimized, multicriteria optimization can be divided into two types:
1. Hierarchical optimization for which if one criteria, say f is more important than other one g, then the problem is to minimize primary criterion f while breaking ties in favor of the schedule that has minimum secondary criterion value g and denoted by Lex(f, g).
2. Simultaneous optimization for which if no criterion is dominant and all criteria are considered simultaneously.

It is well known that the problem 1||\(\sum_j T_j\) is NP-hard [6], so any problem containing the cost function \(\sum_j T_j\) as subproblem is also NP-hard. Sen and Gupta[14] studies the problem of minimizing a linear combination of flow times and maximum tardiness of a given number of jobs on a single machine and presents BAB...
technique to arrive at an optimal solution. The first result involving $E_{\text{max}}$ and $T_{\text{max}}$ is due to Garey et al., they study a single machine scheduling problem and present an O(nlogn) algorithm to check whether for a given threshold value $y$ there exists a feasible schedule such that each job is executed in the interval $[d_i - p_i - y, d_i - y]$. By applying binary search, they find the minimum such value $y$. Garey et al. also show that the problem of finding the set of starting time that minimizes $\sum E_j + \sum T_j$ for a given ordering of the jobs on a single machine is solvable in O(nlogn) time. Verma and Dessouky [15] show that the problem $1\mid p_j = p, \Sigma j(\alpha_i E_j + \beta_i T_j)$ is solvable in polynomial time if the weights are agreeable, that is, the jobs can be renumbered such that: $\alpha_1 \leq \alpha_2 \leq \cdots \leq \alpha_n, \beta_1 \leq \beta_2 \leq \cdots \leq \beta_n$. Abdul Razaq and Ibrahim [1] studied the problem $1\mid |F(E_{\text{max}}, T_{\text{max}})$ they propose a general algorithm (ET) to find the set of approximate efficient (Pareto optimal) solutions, and for the problem $1\mid E_{\text{max}} + T_{\text{max}}$ they find the (near) optimal solution using BAB algorithm. Abdul-Razaq and Mahrooz [2] study problem including three criteria, total completion times, the total tardiness and the maximum tardiness, they propose BAB algorithm for the problem $1\mid \Sigma j C_j + \Sigma j T_j + T_{\text{max}}$ and find the set of efficient solutions for the problem $1\mid \Sigma j C_j, \Sigma j T_j, T_{\text{max}}$. Lee and Choi in [12], propose a genetic algorithm for solving the scheduling problem with distinct due dates in a single machine general penalty weights which are not necessarily proportional to the processing times are applied to jobs either early or tardy. The computational experiments show that the genetic algorithm finds optimal solutions for small instances. Wan and Benjamin in [16] propose a tabu search algorithm for the problem of minimizing the earliness tardiness cost on a single machine with due windows. Computational experiments indicate that the performance of the proposed approach is quite well, especially for the instances of large size.

**Problem formulation**

Let $N = \{1, 2, \ldots, n\}$ be the set of $n$ jobs that must be processed by a machine. Each job $i$ has a processing time $p_i$, and a due date $d_i$, for $i = 1, \ldots, n$. Initially, all of the jobs are available to be processed by the machine and it starts processing without interrupted, and requires $p_i$ units of time to complete its processing. Thus a schedule for the machine can be completely specified by giving the sequence in which the jobs are processed.

Let $\sigma$ be a sequence of the jobs in $N$ represented by the $n$-tuple $(\sigma(1), \sigma(2), \ldots, \sigma(n))$ where $\sigma(i)$ is the $i$th job processed by the machine. The completion time of job $\sigma(i)$ is given by $C_{\sigma(i)} = \sum_{j=1}^{i} p_{\sigma(j)}$. the tardiness of the job $\sigma(i)$ is given by $T_{\sigma(i)} = \max(C_{\sigma(i)} - d_{\sigma(i)}, 0)$, and the earliness of the job $\sigma(i)$ is given by $E_{\sigma(i)} = \max(d_{\sigma(i)} - C_{\sigma(i)}, 0)$.

We consider the following performance criteria:
- the sum of completion times $\sum j C_j$.
- the sum of tardiness $\sum j T_j$.
- the maximum tardiness $T_{\text{max}}$.
- the maximum earliness $E_{\text{max}}$.

Hence this problem is denoted by $1\mid \Sigma j C_j + \Sigma j T_j + T_{\text{max}} + E_{\text{max}}$ (Q) which belong to type (2) of multicriteria optimization and written as:

$$z = \min\left(\sum j C_j + \sum j T_j + T_{\text{max}} + E_{\text{max}}\right)_{S,T}$$

$$C_i \geq p_i \quad i = 1, 2, \ldots, n$$

$$C_i = C_{i-1} + p_j \quad i = 2, 3, \ldots, n$$

$$T_i \geq C_i - d_i \quad i = 1, 2, \ldots, n$$

$$T_i \geq 0 \quad i = 1, 2, \ldots, n$$

$$E_i \geq d_i - C_i \quad i = 1, 2, \ldots, n$$

$$E_i \geq 0 \quad i = 1, 2, \ldots, n$$

**Solving the Problem (Q) Using BAB Algorithm**

The aim of this problem is to find the minimum value of the of the objective function $\sum j C_j + \Sigma j T_j + T_{\text{max}} + E_{\text{max}}$, this function include the cost function $\Sigma j T_j$, as mentioned before any problem including this function as sub problem is NP-hard. Since the feasible set of solutions considered in our problem is finite set, so an
optimal solution can be obtained by a straightforward algorithm that enumerates all feasible solutions, and then outputs the one with the minimum objective value. Complete enumeration method (CEM) is hardly practical because the number of cases to be considered is usually enormous. Solving NP-hard discrete optimization problems to optimality is often an immense job requiring very efficient algorithms, and Branch and bound algorithm (BAB) is one of the main tools in construction of these. A BAB algorithm searches the complete space of solutions for a given problem for the best solution. However, explicit enumeration is normally impossible due to the exponentially increasing number of potential solutions. A BAB algorithm proceeds by repeatedly partitioning the class of all feasible solutions into smaller and smaller subclasses in such a way that ultimately an optimal solution is obtained. The BAB have the following general characteristics:

- A branching rule that defines partitions of the set of feasible solutions into subsets.
- A lower bounding rule that provides a lower bound (LB) on the value of each solution in a subset generated by the branching rule.
- A search strategy that selects a node from which to branch.

To implement the BAB algorithm we decompose the problem (Q) into subproblems:

**Decomposition of the Problem (Q)**

Decomposition is a general approach for solving a problem by breaking it up into smaller ones and solving each of the smaller ones separately, either in parallel or sequentially. The problem (Q) can be decomposed into four subproblems $Q_i$, $i = 1, 2, 3, 4$ : 1) $Z_i$, where $Z_i = \min \sum C_j$, $Z_2 = \min \sum T_j$, $Z_3 \min T_{max}$, $Z_4 = \min E_{max}$.

**Derivation of Lower bound (LB) and Upper bound (UB) for the Problem (Q)**

Next theorem help us to derive the lower bound (LB) for problem (Q) using the decomposed problems:

**Theorem 1:** Let $Z_i$ be the lower bound or the minimum for the objective functions of sub problems $Q_i$, $i = 1, 2, 3, 4$ and let $Z$ be the minimum of objective function of problem Q. Then $\sum Z_i \leq Z$

**Proof:** It is clear

The initial lower bound $ILB = \sum_{i=1}^{n} Z_i$ where $Z_i$ calculated by sequencing the jobs in SPT order to get the minimum value of total completion times. $Z_2$ calculated by sequencing the jobs in EDD order to get the minimum maximum tardiness $T_{max}(EDD)$ using that $T_{max}(EDD) = \sum T_j Z_3$ calculated by sequencing the jobs in EDD order to get minimum maximum tardiness $T_{max}(EDD)$. $Z_4$ calculated by sequencing the jobs in MST order to get the minimum maximum earliness $E_{max}(MST)$. Hence the initial lower bound $ILB$ calculated as follows:

$ILB = \sum_{i=1}^{n} C_i(SPT) + T_{max}(EDD) + T_{max}(EDD) + E_{max}(MST)$

The upper bound (UB) is derived as follows: Compute the first and second upper bound by sequencing the jobs in SPT and EDD order respectively, then

$UB1 = \sum C_i(SPT) + \sum T_j(SPT) + T_{max}(SPT) + E_{max}(SPT)$,

$UB2 = \sum C_i(EDD) + \sum T_j(EDD) + T_{max}(EDD) + E_{max}(EDD)$.

The upper bound is computed by: $UB = \min (UB1, UB2)$.

**Dominance Rule**

A dominance rule is established in order to reduce the solution space either by adding new constraints to the problem, or by writing a procedure that attempts to reduce the domain of variables, or by building interesting solutions directly [10]. This dominance rule is expressed as "There exist at least one optimal solution of $S$ (the set of feasible solutions) having the property A". The strategy is as follows: Any solution which does not satisfy A can be
removed from S, because there is at least one optimal solution of S satisfying A.

**Theorem 2:** If \( p_i \leq p_j \) and \( d_i \leq d_j \) for every \( i, j = 1, 2, ..., n \), then job i precede job j in optimal solution for the problem (Q).

**Proof:** consider the sequence \( \sigma = \sigma_i \sigma_j \sigma_2 \) and the sequence \( \hat{\sigma} = \sigma_1 \sigma_j \sigma_2 \) which is obtained by interchange the position of jobs i and j.

For the sequence \( \sigma \) and \( \hat{\sigma} \) there are two cases

**Case 1:** If \( p_i \leq p_j \) and \( d_i \leq d_j \) implies \( s_i \leq s_j \) for every \( i, j = 1, 2, ..., n \)

From \( p_i \leq p_j \) we have : \( \Sigma_j C_j(\sigma) \leq \Sigma_j C_j(\hat{\sigma}) \)

From the condition of slack time \( s_i \leq s_j \), we have:

\[
E_{\text{max}}(\sigma) \leq E_{\text{max}}(\hat{\sigma}).
\]

From \( p_i \leq p_j \) and \( d_i \leq d_j \), we have:

\[
T_{\text{max}}(\sigma) \leq T_{\text{max}}(\hat{\sigma}) \text{ and } \Sigma_j T_j(\sigma) \leq \Sigma_j T_j(\hat{\sigma}).
\]

Hence we have:

\[
\Sigma_j C_j(\sigma) + \Sigma_j T_j(\sigma) + T_{\text{max}}(\sigma) + E_{\text{max}}(\sigma) \leq \Sigma_j C_j(\hat{\sigma}) + \Sigma_j T_j(\hat{\sigma}) + E_{\text{max}}(\hat{\sigma}).
\]

**Case 2:** If \( p_i \leq p_j \) and \( d_i \leq d_j \) implies \( s_i \geq s_j \) for every \( i, j = 1, 2, ..., n \)

From \( p_i \leq p_j \) we have:

\[
\Sigma_j C_j(\sigma) \leq \Sigma_j C_j(\hat{\sigma}) \quad (1)
\]

The condition on processing times insure that (1) is satisfied, and the addition in cost which is obtained from (1) is equal to \( p_j - p_i \), this gives:

\[
\Sigma_j C_j(\sigma) + p_j - p_i = \Sigma_j C_j(\hat{\sigma}) \quad (2)
\]

From the condition of slack times \( s_i \geq s_j \) implies \( E_{\text{max}}(\sigma) \geq E_{\text{max}}(\hat{\sigma}) \).

Also the addition in cost \( s_i - s_j \) gives:

\[
E_{\text{max}}(\hat{\sigma}) + s_i - s_j = E_{\text{max}}(\sigma) \quad (3)
\]

\[
s_i - s_j = (d_i - p_i) - (d_j - p_j) \leq p_j - p_i \quad (4)
\]

Adding \( E_{\text{max}}(\hat{\sigma}) \) to both side of (4) we have:

\[
E_{\text{max}}(\hat{\sigma}) + s_i - s_j \leq E_{\text{max}}(\hat{\sigma}) + p_j - p_i \text{ and from (3) we have:}
\]

\[
E_{\text{max}}(\sigma) \leq E_{\text{max}}(\hat{\sigma}) + p_j - p_i \quad (5)
\]

Adding \( \Sigma_j C_j(\sigma) \) to both side of (5) and by (2) we have:

\[
\Sigma_j C_j(\sigma) + E_{\text{max}}(\sigma) \leq \Sigma_j C_j(\hat{\sigma}) + E_{\text{max}}(\hat{\sigma}) \quad (6)
\]

From the conditions \( p_i \leq p_j \) and \( d_i \leq d_j \) we have:

\[
T_{\text{max}}(\sigma) \leq T_{\text{max}}(\hat{\sigma}) \text{ and } \Sigma_j T_j(\sigma) \leq \Sigma_j T_j(\hat{\sigma}).
\]

By adding this result to (6):

\[
\Sigma_j C_j(\sigma) + \Sigma_j T_j(\sigma) + T_{\text{max}}(\sigma) + E_{\text{max}}(\sigma) \leq \Sigma_j C_j(\hat{\sigma}) + \Sigma_j T_j(\hat{\sigma}) + E_{\text{max}}(\hat{\sigma})
\]

Hence in both cases the sequence \( \sigma \) is better than the sequence \( \hat{\sigma} \). Hence job i precede job j in the optimal solution.

**Proposition 1:** If SPT and EDD rules are identical in one sequence, then this sequence gives optimal solution for problem (Q).

**Proof:** It is clear from theorem (2).

**Solving the Problem (Q) Using Local Search Algorithms (LSAs)**

Branch and bound algorithm (and dynamic programming) is based on the idea of intelligently enumerating all feasible solutions. Another possibility is to apply (LSAs). These algorithms produce solutions that are guaranteed to be within a fixed percentage of the actual optimum. One of the most successful methods of attacking hard combinatorial optimization problems is the discrete analog of “hill climbing”, known as local (or neighborhood) search [4]. In neighborhood search, a current solution is transformed into a new solution according to some neighborhood
structure. An acceptance rule decides whether the move from the current solution to the transformed solution should be accepted, although the decision is sometimes delayed until the complete neighborhood (or a subset of it) is explored. If a move is accepted, then the transformed solution replaces the previous solution and becomes the current solution; otherwise, the move is rejected and the current solution is retained. This process is repeated until some termination criterion is satisfied. The acceptance rule is usually based on the objective function values of the current solution and its neighbor.

**Descent Method (DM)**

The simplest type of local search algorithms is descent method DM, which is sometimes known as iterative local improvement. In this method, only moves that result in an improvement in the objective function value are accepted. Under a first improve search, the first move that improves the objective function value is accepted [3]. On the other hand, best improve selects a move that yields the best objective function value among all neighbors. When no further improvement can be achieved, a DM terminates with a solution that is a local optimum. The local optimum is not necessarily the true global optimum.

**Simulated Annealing (SA)**

Simulated annealing is a local search algorithm (metaheuristics) capable of escaping from local optima ease of implementation and convergence properties. At each iteration of SA, the values for two solutions (the current solution and a newly selected solution) are compared. Improving solutions are always accepted; while a fraction of non-improving (inferior) solutions are accepted in the hope of escaping local optima in search of global optima. The probability of accepting non-improving solutions depends on a temperature parameter, which is typically non-increasing with each iteration of the algorithm [13].

**Tabu Search (TS)**

Tabu search is another example of neighborhood search that, like SA, is capable of avoiding being trapped in local optima, but the operation of TS is quite different than SA's. At each iteration of TS, a subset of the neighbors of the current solution is considered, and a best of these is chosen. This contrasts to SA which chooses a neighboring solution at random and accepts or rejects it on the basis of a probabilistic function. The subset of neighboring solution considered at each step is made up of all the solutions in the neighborhood minus some set of solutions which are considered tabu. The tabu solutions (tabu list) are usually solutions or moves that would bring the search back to a solution that has already been visited[9]. The tabu list is a form of short term memory that guides the search away from areas that have already been seen.

**Simple Heuristic method (SH)**

We use the Simple Heuristic method (SH) to find initial solutions for the local search algorithms [7], which is the following steps:

**Step 1:** order the jobs in SPT rule.

**Step 2:** set k=2, chose the first two jobs in ordered sequence, schedule them in order to minimize the objective function, and set the better one as the current solution.

**Step 3:** Increment k by 1, and generate k candidate sequences by inserting the first job in the remaining jobs into each slot of the current solution, select the best one from these solutions that minimize the objective function. Update the selected partial solution as the new current solution.

**Step 4:** If k=n. stop, otherwise go to step 3.

**Materials and Methodology**

**Computational Experiments:**

**Test problems:** All experimental design tests are conducted on a personal computer intel (R) Core TM i7 CPU @ 2.50 GHz, and 8.00 GB of RAM. To present the efficiency and compared the results for the problem(Q). Instances with different sizes are considered. The processing times \( p_i, i = 1,2,\ldots,n \) for each problem is generated randomly from uniform distribution on the interval \([1, 10]\), the due dates \( d_i, i = 1,2,\ldots,n \) are considered, which are generated randomly from uniform distribution on the interval \([-10, 10]\).
1,2,...,n is drown from uniform distribution on \( \left[ \left( 1 - \text{TF} - \frac{\text{RDD}}{2} \right) t, \left( 1 - \text{TF} + \frac{\text{RDD}}{2} \right) t \right] \), where \( t \) is total processing times for all jobs. The valueRDD is the relative range of due dates, it determines the length of the interval from which the due dates were taken, TF the tardiness factor determines the relative positions of the center of this interval between 0 and \( t \), these value of TF and RDD are chosen from the set \( \{0.2,0.4,0.6,0.8,1.0\} \). From 25 pairs of values of TF and RDD we generate 10 problem instances for each \( n \) from \( n=4 \) to \( n=25 \) for BAB algorithm, while for the LSAs from \( n=50 \) to \( n=2000 \) for large size problems.

Results and Discussions

Comparison of the results of the problem (Q) using BAB algorithm:

Table 1 shows the average of experimental results of the two methods BAB without DR and BAB with DR. This table contains the number of jobs (\( n \)), for each of the ten instances, the average number of nodes, the average computational times, the average of optimal values, and the percentage of solved instances for problem(Q). We see that the BAB with DR gives fairly good results in terms of computational times and the number of nodes. For example for \( n=16 \) the BAB without DR failed to find optimal solution within 1800 second for some instances and solved 60 % of these instances, while BAB with DR solved problems in average CPU Time = 2.16 seconds and average number of nodes = 46267.9. Also Table (1) shows that the BAB without DR solved all problem instances from \( n=4 \) to \( n=14 \), and failed to solve all problems from \( n=21 \) to \( n=25 \), while the BAB with DR solved the problem in all of instances from \( n=4 \) to \( n=23 \) and failed to solve problem in one problem when \( n=24 \), and two problems when \( n=25 \).

<table>
<thead>
<tr>
<th>n</th>
<th>BAB without DR</th>
<th>BAB with DR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean Best</td>
<td>Mean Nods</td>
</tr>
<tr>
<td>4</td>
<td>55.5</td>
<td>11.5</td>
</tr>
<tr>
<td>5</td>
<td>97</td>
<td>22.9</td>
</tr>
<tr>
<td>6</td>
<td>110.7</td>
<td>52.7</td>
</tr>
<tr>
<td>7</td>
<td>151.5</td>
<td>137</td>
</tr>
<tr>
<td>8</td>
<td>195.3</td>
<td>373.2</td>
</tr>
<tr>
<td>9</td>
<td>255.3</td>
<td>1370.9</td>
</tr>
<tr>
<td>10</td>
<td>283.5</td>
<td>4383.3</td>
</tr>
<tr>
<td>11</td>
<td>349.5</td>
<td>16626.2</td>
</tr>
<tr>
<td>12</td>
<td>396.5</td>
<td>45324.6</td>
</tr>
<tr>
<td>13</td>
<td>426.9</td>
<td>207839.2</td>
</tr>
<tr>
<td>14</td>
<td>529.1</td>
<td>3213032</td>
</tr>
<tr>
<td>15</td>
<td>696.3</td>
<td>11636173</td>
</tr>
<tr>
<td>16</td>
<td>693.1</td>
<td>20610685</td>
</tr>
<tr>
<td>17</td>
<td>716.3</td>
<td>21511161</td>
</tr>
<tr>
<td>18</td>
<td>851.4</td>
<td>41939475</td>
</tr>
<tr>
<td>19</td>
<td>967.5</td>
<td>43225118</td>
</tr>
<tr>
<td>20</td>
<td>928.8</td>
<td>38241525</td>
</tr>
<tr>
<td>21</td>
<td>1154.7</td>
<td>45789608</td>
</tr>
<tr>
<td>22</td>
<td>1075.4</td>
<td>43871985</td>
</tr>
<tr>
<td>23</td>
<td>1285.9</td>
<td>43273922</td>
</tr>
<tr>
<td>24</td>
<td>1336.4</td>
<td>44816660</td>
</tr>
<tr>
<td>25</td>
<td>1500.2</td>
<td>43181493</td>
</tr>
</tbody>
</table>
Comparison of results of the problem (Q) using local search algorithms

Computational results of (LSAs), (SA), (TS), and (DM) summarized in tables (2), (3). For the implementation of LSAs we generate the initial solution by SH algorithm which described in section 4.4, and we use arrange of number of iterations starts from 20000 iteration for small size problems where $4 \leq n \leq 23$, and use 50000 iterations for medium size problems where $50 \leq n \leq 400$, for large size problems where $n \geq 500$ we use 100000 iterations. The neighborhoods generated using two methods, the API (Adjacent Pairwise Interchange) method and insertion method, and the algorithm interchange between the two neighborhoods at each iteration according to the number of iterations is odd or even. The comparison between BAB algorithm and LSAs is summarized in Table 2, the results shows that SA and TS algorithms solve all problems and reach the optimal solutions for all small size problems where DM algorithm is not for some instances, also the results shows that TS need more time than other LSAs to reach the optimal solutions. In Table 3 we summarize the results of comparison among LSAs themselves, for each algorithm we find the mean of best values and the mean of computation times. Also Table 3 shows that the performance of SA and TS algorithms is better than DM and that SA algorithm better than TS algorithm.

<table>
<thead>
<tr>
<th>n</th>
<th>BAB</th>
<th>SA</th>
<th>TS</th>
<th>DM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean optimal</td>
<td>Mean Time</td>
<td>Mean value</td>
<td>Mean Time</td>
</tr>
<tr>
<td>4</td>
<td>55.5</td>
<td>0.0170</td>
<td>55.5</td>
<td>0.6663</td>
</tr>
<tr>
<td>5</td>
<td>97</td>
<td>0.0008</td>
<td>97</td>
<td>0.7883</td>
</tr>
<tr>
<td>6</td>
<td>110.7</td>
<td>0.0009</td>
<td>110.7</td>
<td>0.6472</td>
</tr>
<tr>
<td>7</td>
<td>151.5</td>
<td>0.0016</td>
<td>151.5</td>
<td>0.6544</td>
</tr>
<tr>
<td>8</td>
<td>195.3</td>
<td>0.0026</td>
<td>195.3</td>
<td>0.6548</td>
</tr>
<tr>
<td>9</td>
<td>255.3</td>
<td>0.0037</td>
<td>255.3</td>
<td>0.6650</td>
</tr>
<tr>
<td>10</td>
<td>283.5</td>
<td>0.0099</td>
<td>283.5</td>
<td>0.6499</td>
</tr>
<tr>
<td>11</td>
<td>349.5</td>
<td>0.0109</td>
<td>349.5</td>
<td>0.7321</td>
</tr>
<tr>
<td>12</td>
<td>396.5</td>
<td>0.0185</td>
<td>396.5</td>
<td>0.6624</td>
</tr>
<tr>
<td>13</td>
<td>426.9</td>
<td>0.0359</td>
<td>426.9</td>
<td>0.7122</td>
</tr>
<tr>
<td>14</td>
<td>529.1</td>
<td>0.3167</td>
<td>529.1</td>
<td>0.6637</td>
</tr>
<tr>
<td>15</td>
<td>696.3</td>
<td>0.7606</td>
<td>696.3</td>
<td>0.6981</td>
</tr>
<tr>
<td>16</td>
<td>693</td>
<td>1.9578</td>
<td>693</td>
<td>0.6631</td>
</tr>
<tr>
<td>17</td>
<td>716.1</td>
<td>0.3007</td>
<td>716.1</td>
<td>0.6602</td>
</tr>
<tr>
<td>18</td>
<td>850.4</td>
<td>3.5619</td>
<td>850.5</td>
<td>0.7051</td>
</tr>
<tr>
<td>19</td>
<td>966.1</td>
<td>11.164</td>
<td>966.1</td>
<td>0.6789</td>
</tr>
<tr>
<td>20</td>
<td>928.3</td>
<td>38.04</td>
<td>928.4</td>
<td>0.6771</td>
</tr>
<tr>
<td>21</td>
<td>1152.7</td>
<td>160.24</td>
<td>1152.7</td>
<td>0.6788</td>
</tr>
<tr>
<td>22</td>
<td>1073.7</td>
<td>198.12</td>
<td>1073.7</td>
<td>0.6822</td>
</tr>
<tr>
<td>23</td>
<td>1282</td>
<td>82.756</td>
<td>1282</td>
<td>0.7448</td>
</tr>
</tbody>
</table>

Table 2: The average results of BAB and local search algorithms for small size problems.
Table 3: The average results of local search algorithms for large size problems.

<table>
<thead>
<tr>
<th>n</th>
<th>SA Mean Best</th>
<th>SA Mean Time</th>
<th>TS Mean Best</th>
<th>TS Mean Time</th>
<th>DM Mean Best</th>
<th>DM Mean Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>5654.7</td>
<td>1.8871</td>
<td>5654.9</td>
<td>4.9127</td>
<td>5656.6</td>
<td>1.8518</td>
</tr>
<tr>
<td>100</td>
<td>20078</td>
<td>2.3228</td>
<td>20079.1</td>
<td>7.6682</td>
<td>20079.9</td>
<td>2.2943</td>
</tr>
<tr>
<td>150</td>
<td>46948.4</td>
<td>2.4242</td>
<td>46946.2</td>
<td>10.604</td>
<td>46957.9</td>
<td>2.4356</td>
</tr>
<tr>
<td>200</td>
<td>81334.5</td>
<td>5.9961</td>
<td>81335.5</td>
<td>73.222</td>
<td>81340.8</td>
<td>6.0744</td>
</tr>
<tr>
<td>250</td>
<td>124500.8</td>
<td>7.4342</td>
<td>124491.8</td>
<td>10.604</td>
<td>124502.1</td>
<td>7.4767</td>
</tr>
<tr>
<td>300</td>
<td>174800.5</td>
<td>8.9283</td>
<td>174795.3</td>
<td>118.92</td>
<td>174809.8</td>
<td>8.8176</td>
</tr>
<tr>
<td>400</td>
<td>312125.8</td>
<td>9.0340</td>
<td>312146.1</td>
<td>101.38</td>
<td>312191.7</td>
<td>8.9497</td>
</tr>
<tr>
<td>500</td>
<td>501303.4</td>
<td>12.290</td>
<td>501319.3</td>
<td>126.61</td>
<td>501410.6</td>
<td>12.209</td>
</tr>
<tr>
<td>600</td>
<td>71688.5</td>
<td>16.435</td>
<td>71678.1</td>
<td>152.47</td>
<td>71698.3</td>
<td>16.372</td>
</tr>
<tr>
<td>700</td>
<td>985236.6</td>
<td>21.683</td>
<td>985328.1</td>
<td>215</td>
<td>985365.5</td>
<td>21.617</td>
</tr>
<tr>
<td>800</td>
<td>1259230.2</td>
<td>28.439</td>
<td>1259180.1</td>
<td>215.35</td>
<td>1259230.7</td>
<td>28.379</td>
</tr>
<tr>
<td>900</td>
<td>1610269.8</td>
<td>36.401</td>
<td>1610361</td>
<td>245.78</td>
<td>1610416</td>
<td>36.297</td>
</tr>
<tr>
<td>1000</td>
<td>1956879.4</td>
<td>46.160</td>
<td>1956764.1</td>
<td>284.86</td>
<td>1956735.2</td>
<td>46.007</td>
</tr>
<tr>
<td>1500</td>
<td>4439021.1</td>
<td>125.85</td>
<td>4439195.6</td>
<td>767.05</td>
<td>4439669.1</td>
<td>125.45</td>
</tr>
<tr>
<td>2000</td>
<td>7868336</td>
<td>272.65</td>
<td>7868393.4</td>
<td>1464.2</td>
<td>7868473.6</td>
<td>272.92</td>
</tr>
</tbody>
</table>

Conclusions

It is well known that the sizes and the results of multicriteria scheduling problems are generally affected by a number of cost functions, and our problem ($Q$) is the sum of four cost functions. Therefore BAB algorithm failed to solve problems up to 25 jobs, on the other hand the comparison between BAB without and with dominance rule shows that dominance rule improve the performance of the BAB algorithm in both computation times and optimal values. In LSAs if we don’t focus our attention on computation times we see that SA is better than TS and DM to solve our problem($Q$). Because of its structure TS need more time than other algorithms to find best solution, but this time remain less than 1800 seconds.

References


[9] J. D. Knowles, "Local-search and hybrid evolutionary algorithms for Pareto...
optimization," University of Reading UK, 2002.


