Ant Colony Optimization on scheduling problem with job values given as a power function of their completion times

Sachin Shrivastava, Deepak Chaudhary, Kapil Shrivastava

Abstract—This paper deals with the problem of scheduling jobs on the identical parallel machines, where job values are given as a power function of the job completion times. Minimization of the total loss of job values is considered as criterion. This paper establishes the computational complexity of the problem by the application of Ant Colony Optimization on the problem. Strong NP hardness of its general versions and NP hardness of its single machine case. Moreover, some special cases of the problem in the polynomial time. Finally the project construct and experimentally test branch and bound algorithm along with some elimination properties improving its efficiency. In order to solve the significant real life problems a lot of new scheduling problems have been formulated. The problems with resource allocation and the problem with deteriorating jobs are two kinds of examples of such scheduling.

Index Terms—Scheduling, Scheduling Algorithm, Parallel Processor, Ant Colony, ACO, MMAS, job values, loss function

1 INTRODUCTION

The aim of this scheduling is to describe and solve a scheduling problem where job values (or losses of jobs values) changes during their execution and are described as a power function of the job completion times. In this paper we introduce a new non decreasing power model of loss of job values and formulate a problem of minimization of the total job values loss which can be used to solve the problem of establishing an order of object renovation. Here we introduce a new model of job value, defined as a difference between initial job value and non decreasing power function loss. [1]

Another new kind of the scheduling problems that we had studied in this paper. Namely the in which processing times of all the jobs are some values which are fixed in advance and constant during optimization process but their values deteriorate over time. The process description of the problem can be illustrated by an application example, which characterizes the utilization process of the components from some used up computers. Therefore assumption that there is given a set of some used up computer which cannot be used any more because their further utilization is connected with a high risk of a breakdown.

There are some application which require faster processors or simply of their components are already broken. However some of their components (e.g. monitors, floppy disks drives, network drives, poor suppliers) can be utilized as spare parts in some other computers. Thus the problem of disassembling computers into their components appears nearest can be shown that the values of the computer component decreases over time as a power function of the speed of this different for the particular component. In disassembling process we are interested in the component values which are determined at the moments at they are available for utilization. The component is ready to be used after it is completely removed from one computer and its proper functionality is confirmed. Thus the order in which the computer will be disassembled has a significant influenced on the total profit, i.e. the sum of the component values. Therefore, maximization of the total component value is considered as an optimization criterion [2]

2 SCHEDULING

A schedule is a tangible plan or document, such as a bus or a class schedule. A schedule usually tells us when things are supposed to happen; it shows us a plan for the timing of certain activities and answers the question, “If all goes well, when will a particular event take place?”

The aim of this scheduling is to describe and solve a scheduling problem where jobs values (or losses of jobs values) changes during their execution and are described as a power function of the job completion times. [3]

2.1 Notations

Regardless of its nature, every scheduling problem S can be formulated as a quadruple, S = (J, M, P, L), where J is a set of non preemptive jobs immediately available for processing time 0. M is a set of identical parallel machines entities that will perform the available jobs. P is the processing time of the job, where pj > 0 and L is the loss function as lj(t) characterizing the loss of its value at time t. [4]

Processing time (pij) - The pij represents the processing time of job j on machine i. The subscript i is omitted if the processing time of job j does not depend on the machine or if job j
is only to be processed on one given machine.

**Exponential loss rate (aj) -** This is the loss rate denoted by the aj and aj > 0 and it is associated with the job ji performing on machine M.

**Proportional loss rate (wj) -** This is another loss rate denoted by the wj and wj > 0 and associated with the job ji performing on machine M.

**Loss function (l(j,t)) -** This is loss rate function characterizing the loss of its value at time t. The loss rate function is a non-decreasing power function of time. Moreover, the loss rate is calculated only at the completion time Cj.

Three basic pieces of information that help to describe jobs in the single-machine case are:
- Processing time (pj) the amount of processing required by job j
- Exponential loss rate (aj) the exponential loss rate of job j
- Proportional loss rate (wj) The proportional loss rate of job j
- Completion time (Cj) The time at which the processing of job j is finished.

### 2.2 Scheduling Algorithm and its Complexity

A useful perspective on the relation of scheduling problems and their solution techniques comes from developments in a branch of computer science known as complexity theory. The notion of complexity refers to the computing effort required by a solution algorithm. Computing effort is described by order-of-magnitude notation. For example, suppose we use a particular algorithm to solve a problem of size n. (Technically, n denotes the amount of information needed to specify the problem.) The number of computations required by the algorithm is typically bounded from above by a function of n. If the order of magnitude of this function is polynomial, as n gets large, then we say the algorithm is polynomial. For instance, if the function has order of magnitude n^2, denoted O(n^2), then the algorithm is polynomial. On the other hand, if the function is O(2^n), then the algorithm is non-polynomial (in this case, exponential). Other things being equal, we prefer to use a polynomial algorithm because as n grows large, polynomial algorithms are ultimately faster. [5]
by the user. And we generate the sequence number with processing time.

And secondly we generate another sequence number by assigning the jobs to the machines on the basis of their loss function. The loss function is denoted by \( l_j(t) \) characterizing the loss of its value at time 0. The loss function is denoted by non decreasing power function. [6]

For this section we describe the most efficient algorithms and the result of the performed experimentally analysis. All the algorithms A1LST-A3LST are based on the list strategy but they use three different priority dispatching rules to obtain an order of jobs in an input list. [7]

### 4.1 Algorithm A1LST

Step 1: Construct the list of jobs \( L \), according to the algorithm A1, \( i \in \{1, 2, 3\} \).

Algorithm A1 and A2 construct solutions by sequencing all the jobs in the non-increasing order of \( a_i \) and \( (a_i \cdot w_i / p) \) respectively.

#### Algorithm A3

Step 1: Set \( \pi = 0, U = \{1, \ldots, n\} \) and \( C = \sum_{i=1}^{n} p_i \).

Step 2: Find a job \( j \in U \) with the minimal value of \( w \cdot C \) and move it from \( U \) at the beginning of \( \pi \). Set \( C = C - p_j \).

Step 3: If \( U \neq \emptyset \) then go to step 2; the job order is given by \( \pi \).

The computational complexity of algorithm A1 and A2 is equal to \( O(n^2) \). Finally in order to improve the job sequence on the machines the following procedure is launched when the jobs are scheduled. [8] [9]

#### 4.1.1 Algorithm DoubleSwap

**Step 1:**

For \( j = 1, \ldots, n - 1 \) swap jobs in positions \( j \) and \( j + 1 \) if

\[
1_i^{(1)} (C_{n_i^{(1)}}) + 1_i^{(1)} (C_{n_i^{(1)+1}}) > 1_i^{(1)} (C_{n_i^{(1)+1}}) + 1_i^{(1)} (C_{n_i^{(1)+1}})
\]

**Step 2:**

For \( j = n, \ldots, 2 \) swap jobs in positions \( j \) and \( j - 1 \) if

\[
1_i^{(1)} (C_{n_i^{(1)}}) + 1_i^{(1)} (C_{n_i^{(1)-1}}) > 1_i^{(1)} (C_{n_i^{(1)-1}}) + 1_i^{(1)} (C_{n_i^{(1)-1}})
\]

### 5 ANT ALGORITHM

#### 5.1 Algorithm used in ACO

There is a pool of jobs having constant processing time \( (p_{rt}) \) and initial job value \( (C_0) \) but varying deterioration \( (d_i) \) depending on completion time. The algorithm for parallel processor scheduling using Ant Colony Optimization may be represented as follows:

1. Initialize the pheromone \( \rho_{ij} \) with initial pheromone value \( \tau_0 \cdot \rho \).

Pheromone \( \rho_{ij} \) is accumulated between jobs and deterioration interval. [10]

2. All ants start making its tour.

   a) Ants select a job first. Selection of job depends on two parameter- (1) processing time and (2) next deterioration, which is drive by a probability distribution function. Job \( j \) is selected for deterioration interval \( i \) according to formula [11]

\[
j = \max (p_{rij}, i^{\ast} d_{i\ast}/p_{rtj}) \quad \text{if } q < q_0
\]

\[
j = p(0) = (p_{rij} \cdot d_{i\ast}/p_{rtj}) / \sum (p_{rij} \cdot d_{i\ast}/p_{rtj}) \quad \text{otherwise}
\]

   Where, \( d_i \) = deterioration for interval \( i \)

   \( \rho_{rtj} \) = processing time for job \( j \)

   b) Ants select a processor (p) for selected job (j) according to formula

\[
p = \max [ (t_{pc} - t_{pi}) / (t_{pc} - t_{pi-1}) / i_p]
\]

   Where, \( t_{pc} \) = completion time on processor \( p \)

   \( t_{pi} \) = interval for processor \( p \)

   c) Update the pheromone (local updation)

\[
\rho_{ijp} = \rho_{ijp} + \rho \cdot \tau_0
\]

   d) Steps i to iii is repeated for all jobs.

3. Select the best tour among tours of all ants.

   a) Best tour with maximum job value is selected.

   b) Update the pheromone (global updation)

\[
\rho_{ijp} = (1 - \rho) \cdot \rho_{ijp} + \rho \cdot \tau_0
\]

   where, \( D \) = total deterioration in best tour

   Repeat steps 1 to 3 for each iteration

#### 5.2 Steps for Solve the problem by ACO

The basic steps, which are to be followed for solving the problem by ACO, are as follows [12]

- Represent the problem in the form of sets of components and transitions, or by a set of weighted graphs, on which ants can build solutions
- Define the meaning of the pheromone trails
- Define the heuristic preference for the ant while constructing a solution
- If possible implement a efficient local search algorithm for the problem to be solved
- Choose a specific ACO algorithm and apply to problem being solved
- Tune the parameter of the ACO algorithm

#### 5.3 Ant System

Ant System is the first ACO algorithm proposed in the literature. Its main characteristic is that, at each iteration, the pheromone values are updated by all the m ants that have built a solution in the iteration itself. The pheromone \( \tau_{ijp} \) associated
with the edge joining cities i and j, is updated as follows:[13]

\[ \tau_{ij} \leftarrow (1-\rho) \cdot \tau_{ij} + \sum_{k=1}^{m} \Delta \tau_{ij}^k \]

where \( \rho \) is the evaporation rate, \( m \) is the number of ants, and \( \Delta \tau_{ij}^k \) is the quantity of pheromone laid on edge \((i, j)\) by ant \( k \):

\[ \Delta \tau_{ij}^k = \begin{cases} Q/L_k & \text{if any } k \text{ used edge}(i, j) \text{ in its tour,} \\ 0 & \text{otherwise} \end{cases} \]

where \( Q \) is a constant, and \( L_k \) is the length of the tour constructed by ant \( k \). In the construction of a solution, ants select the following city to be visited through a stochastic mechanism. When ant \( k \) is in city \( i \) and has so far constructed the partial solution \( s_p \), the probability of going to city \( j \) is given by

\[ p_{ij}^k = \begin{cases} \frac{\min(\pi_{ij}, 1)}{\sum_{l \in N(S^p)} \min(\pi_{il}, 1)} & \text{if } c_{ij} \in N(S^p) \\ 0 & \text{otherwise} \end{cases} \]

where \( N(S^p) \) is the set of feasible components; that is, edges \((i, j)\) where \( i \) is a city not yet visited by the ant \( k \). The parameters \( \alpha \) and \( \beta \) control the relative importance of the pheromone versus the heuristic information \( \pi_{ij} \), which is given by:

\[ \pi_{ij} = \frac{1/d_{ij}}{\sum_{l \in N(S^p)} 1/d_{il}} \]

5.3.1 MIN-MAX Ant System (MMAS)

This algorithm is an improvement over the original Ant System. Its characterizing elements are that only the best ant updates the pheromone trails and that the value of the pheromone is bound. The pheromone update is implemented as follows: [15]

\[ \tau_{ij} \leftarrow [(i - \rho) \cdot \tau_{ij} + \Delta \tau_{ij}^\text{best}] \tau_{\text{min}} \]

where \( \tau_{\text{max}} \) and \( \tau_{\text{min}} \) are respectively the upper and lower bounds imposed on the pheromone; the operator \( [x]_{\alpha}^\beta \) is defined as:

\[ [x]_{\alpha}^\beta = \begin{cases} a & \text{if } x > a, \\ b & \text{if } x < b, \\ x & \text{otherwise} \end{cases} \]

And \( \Delta \tau_{ij}^\text{best} \) is:

\[ \Delta \tau_{ij}^\text{best} = \begin{cases} 1/L_{\text{best}} & \text{if } (i, j) \text{ belongs to best tour}, \\ 0 & \text{otherwise} \end{cases} \]

where \( L_{\text{best}} \) is the length of the tour of the best ant. This may be (subject to the algorithm designer decision) either the best tour found in the current iteration—iteration-best, \( L_{ib} \)—or the best solution found since the start of the algorithm—best-so-far, \( L_{bs} \)—or a combination of both. Concerning the lower and upper bounds on the pheromone values, \( \tau_{\text{min}} \) and \( \tau_{\text{max}} \) they are typically obtained empirically and tuned on the specific problem considered. Nonetheless, some guidelines have been provided for defining \( \tau_{\text{min}} \) and \( \tau_{\text{max}} \) on the basis of analytical considerations. [16]

5.4 Ant Colony System

The most interesting contribution of ACS is the introduction of a local pheromone update in addition to the pheromone update performed at the end of the construction process (called offline pheromone update). The local pheromone update is performed by all the ants after each construction step. Each ant applies it only to the last edge traversed: [17]

\[ \tau_{ij} \leftarrow (1 - \phi) \cdot \tau_{ij} + \phi \cdot \tau_{0} \]

where \( \phi \in [0, 1] \) is the pheromone decay coefficient, and \( \tau_{0} \) is the initial value of the pheromone.

The main goal of the local update is to diversify the search performed by subsequent ants during iteration: by decreasing the pheromone concentration on the traversed edges, ants encourage subsequent ants to choose other edges and, hence, to produce different solutions. This makes it less likely that several ants produce identical solutions during one iteration. The offline pheromone update, similarly to MMAS, is applied at the end of each iteration by only one ant, which can be either the iteration-best or the best-so-far. However, the update formula is slightly different:

\[ \tau_{ij} \leftarrow \begin{cases} (1 - \rho) \cdot \tau_{ij} + \rho \cdot \Delta \tau_{ij} & \text{if } (i, j) \text{ belongs to best tour,} \\ \tau_{ij} & \text{otherwise} \end{cases} \]

As in MMAS, \( \Delta \tau_{ij} = 1/L_{\text{best}} \), where \( L_{\text{best}} \) can be either \( L_{ib} \) or \( L_{bs} \). Another important difference between ACS and AS is in the decision rule used by the ants during the construction process. [18][19]

In ACS, the so-called pseudorandom proportional rule is used: the probability for an ant to move from city \( i \) to city \( j \) depends on a random variable \( q \) uniformly distributed over \([0,1] \), and a parameter \( q_0 \) if \( q \leq q_0 \), then

\[ p_{ij}^k = \begin{cases} \sum_{l \in N(S^p)} \min(\pi_{il}, 1) & \text{if } c_{ij} \in N(S^p) \\ 0 & \text{otherwise} \end{cases} \]

6 Results

This paper solves the problem of parallel processor scheduling. It is analyzed that the final results of parallel processor scheduling using ant colony optimization technique are better than the results of heuristics that have been already developed earlier.

6.1 Results of Heuristics

Table 6.1 Result Of Scheduling For 100 jobs
7.1 Resultant graph of multi processor scheduling for 100 jobs

Table 6.2 Result Of Scheduling For 200 jobs

<table>
<thead>
<tr>
<th></th>
<th>VALUE 1</th>
<th>VALUE 2</th>
<th>VALUE 3</th>
<th>VALUE 4</th>
<th>VALUE 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>A2</td>
<td>8.8989</td>
<td>7.8976</td>
<td>8.9907</td>
<td>5.4532</td>
<td>6</td>
</tr>
<tr>
<td>A3</td>
<td>3.2412</td>
<td>2.1342</td>
<td>4.3321</td>
<td>2.1311</td>
<td>4.6754</td>
</tr>
<tr>
<td>DA1</td>
<td>9.8765</td>
<td>7.8564</td>
<td>7.3521</td>
<td>8.3409</td>
<td>6.4409</td>
</tr>
<tr>
<td>DA2</td>
<td>7.8976</td>
<td>6.5643</td>
<td>5.3421</td>
<td>7.3452</td>
<td>8.9856</td>
</tr>
<tr>
<td>DA3</td>
<td>3.2143</td>
<td>2.1045</td>
<td>3.1232</td>
<td>1.5643</td>
<td>4.6867</td>
</tr>
<tr>
<td>ACS1</td>
<td>2.1321</td>
<td>2.1001</td>
<td>1.9987</td>
<td>1.3342</td>
<td>3.0934</td>
</tr>
</tbody>
</table>

* RED COLOR: MAXIMUM LOSS VALUE
* SKY BLUE COLOR: MINIMUM LOSS VALUE

Table 6.3 Result Of Scheduling For 300 jobs

<table>
<thead>
<tr>
<th></th>
<th>VALUE 1</th>
<th>VALUE 2</th>
<th>VALUE 3</th>
<th>VALUE 4</th>
<th>VALUE 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>A2</td>
<td>6.7654</td>
<td>6.7888</td>
<td>7.5643</td>
<td>7.5643</td>
<td>7.3421</td>
</tr>
<tr>
<td>A3</td>
<td>5.6754</td>
<td>3.2213</td>
<td>5.6543</td>
<td>4.5632</td>
<td>4.5621</td>
</tr>
<tr>
<td>DA1</td>
<td>8.8945</td>
<td>8.6554</td>
<td>8.7865</td>
<td>9.7854</td>
<td>8.4654</td>
</tr>
<tr>
<td>DA2</td>
<td>9.1123</td>
<td>6.3452</td>
<td>5.6754</td>
<td>7.7644</td>
<td>5.6754</td>
</tr>
<tr>
<td>DA3</td>
<td>4.4453</td>
<td>3.2218</td>
<td>4.3211</td>
<td>2.1068</td>
<td>3.5632</td>
</tr>
<tr>
<td>ACS1</td>
<td>4.4005</td>
<td>2.4321</td>
<td>3.3305</td>
<td>2.1005</td>
<td>2.5612</td>
</tr>
</tbody>
</table>
7.2 Resultant graph of multi processor scheduling for 200 jobs

7.3 Resultant graph of multi processor scheduling for 300 jobs

7 Conclusion

In this paper the parallel machines scheduling problem with time dependent losses of job values has been investigated. It has been shown that the general version of the problem is strongly NP-hard and its single machine case is NP-hard. Therefore, in order to solve the problem the use of branch and bound algorithm, supported by some elimination properties which improve its efficiency, delivers optimal solution of the instances of 25 jobs in a few minutes on the average, using some heuristics algorithm with low computational complexities – O(nlogn) and O(n²). The experimentally analysis has revealed, that their efficiencies strongly depend on the instance size and the values of the problem parameters. However, the analysis has delivered also a manner of choosing an appropriate algorithm in order to obtain near solutions.

At last comparison is made with ant colony algorithms and we conclude that ant colony gives more optimized result as compare to heuristics.

REFERENCES

[6] Ant Colony optimization Marco Dorigo, Mauro Birattari, and Thomas Stutzle Université Libre de Bruxelles
[10] Cheng T.C.E., Liu, Parallel machine scheduling to minimize the sum of quadratic completion times.
[12] Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem by Marco Dorigo, Senior Member, IEEE, and Luca Maria Gambardella, Member, IEEE.