Implementation Methodology of Real-Valued Augmented Simulated Annealing Algorithm for Dependent Task Scheduling

S. Selvi,

Associate Professor, Department of Electronics and Communication Engineering, Dr. Sivanthi Aditanar College of Engineering, Tamilnadu, India.

ABSTRACT:

As computing resources are distributed in multiple domains in the Internet, the computational and storage nodes and the underlying networks connecting them are heterogeneous. Thus the heterogeneity results in different capabilities for job processing and data access. To achieve the promising potentials of tremendous distributed resources, effective and efficient scheduling algorithms are fundamentally important. Real-valued Augmented Simulated Annealing (RASA) effectively exploits the essentials of Genetic Algorithm (GA) together with the basic concept of simulated annealing method guiding the search towards minimal energy states. This paper explains the methodology of application of RASA algorithm for the constrained task scheduling problems.

Keywords: Real-valued Augmented Simulated Annealing, Constrained Task Scheduling, DAG, Makespan, Ranking

INTRODUCTION:

A task scheduling is the mapping of tasks to a selected group of resources which may be distributed in multiple administrative domains. A scheduling problem is specified by a set of machines, a set of jobs/operations, optimality criteria, environmental specifications, and by other constraints[1]. Given an application modeled by the Directed Acyclic Graph (DAG), the scheduling problem deals with mapping each task of the application onto the available heterogeneous systems in order to minimize makespan [2]. DAG includes the characteristics of an application program such as the execution time of tasks, the data size to communicate between tasks and task dependencies. The task scheduling problem has been solved several years ago and is known to be NP-complete [3,4].

In general, task scheduling algorithm for heterogeneous systems is classified into two classes: static and dynamic [5]. In static scheduling algorithms, all information needed for scheduling must be known in advance [6]. Static task scheduling takes place during compile time before running the parallel application. In contrast, scheduling decisions in dynamic scheduling algorithms are made at run time.

PROBLEM DEFINITION:

The task scheduling problem is the process of assigning a set of \( v \) tasks in a DAG to a set of \( q \) computing nodes, which have diverse characteristics, without violating the precedence constraints. Before scheduling, the priority of execution of tasks is calculated based on the upward ranking methodology [4]. The tasks are sorted in the decreasing order of the upward rank value. The highest priority task (with high rank value), has the highest scheduling
priority. If more than one task has equal upward rank value, the scheduling priority of the task is decided randomly.

In this paper, the schedule length of the given DAG application, namely makespan, is the largest finish time among all tasks, which is the actual finish time of the exit task, \( n_{exit} \). The objective of the task scheduling problem is to minimise the makespan (fitness), without violating the precedence constraints of the tasks. The objective function is defined in Equation (1) [4].

\[
fitness = Makespan = f(x) = \begin{cases} 
EFT(n_{exit}), & \text{for single } n_{exit} \\
\max\{EFT(n_{exit})\}, & \text{for multiple } n_{exit}
\end{cases} 
\]  

(1)

where EFT is the Earliest Finish Time of the task \( n_i \) on the computing node \( p_j \), defined in the Equation (2) [4].

\[
EFT(n_i, p_j) = w_{i,j} + EST(n_i, p_j) 
\]  

(2)

where \( EST(n_i, p_j) \) is the Earliest Start Time of the task \( n_i \) on the computing node \( p_j \), defined in the Equation (3) [4].

\[
EST(n_i, p_j) = \begin{cases} 
\max\{avail\_time(p_j), ready\_time(n_i)\}, & \text{if } n_i \neq n_{entry} \\
0, & \text{if } n_i = n_{entry}
\end{cases} 
\]  

(3)

where \( avail\_time(p_j) \) is the earliest time at which the computing node \( p_j \) is ready for the task execution and \( ready\_time(n_i) \) is the time when all data needed by \( n_i \) has arrived at the computing node \( p_j \), defined in the Equation (4) [4].

\[
ready\_time(n_i) = \max_{n,m \in pred(n_i)} (EFT(n_m) + c_{m,i}) 
\]  

(4)

where \( pred(n_i) \) is the set of predecessor tasks of the task \( n_i \).

RASA ALGORITHM

The augmented simulated annealing method is the combination of two stochastic optimization techniques-genetic algorithm and simulated annealing. This method effectively exploits the essentials of genetic algorithm together with the basic concept of simulated annealing method, guiding the search towards minimal energy states. Real-valued augmented simulated annealing makes use of the replacement procedure which is controlled by Metropolis criterion. The algorithm 1 describes an implementation of the RASA[7].

Algorithm 1: Real-valued Augmented Simulated Annealing Algorithm

0) Initialize \( T_{max} \), countermax, successmax
1) \( T_t = T_{max}, t = 0 \)
2) Generate \( P \), evaluate \( P \)
3) **while** (not termination condition)  
4) counter = success= 0  
5) **while** (counter < countermax ^ success < successmax)  
6) counter = counter +1, t = t +1  
7) select mutation operator O  
8) select individuals I_t from P_t  
9) modify I_t by O to generate I'_t  
10) p = exp((F(I) - F(I'))/T_t)  
11) if (u(0,1) ≤ p)  
12) insert I'_t into P_t  
13) **end**  
14) success = success+1  
15) evaluate P_t  
16) **end**  
17) decrease T_t  
18)**end**

Where F ( . ) be the fitness of the individual.

4.1 **List of mutation operators**  
The following set of real-valued operators, proposed in [23] is used for the implementation of RASA. In the sequel, we will denote L and U as vectors of lower/upper bounds on unknown variables, u (a, b) and u [a, b] as a real or integer random variable with the uniform distribution on a closed interval <a, b>.

4.1.1 **Uniform mutation**  
Let k = [1, n]  
\[ ch_{ij}(t + 1) = \begin{cases} 
  u(L_j, U_j), & \text{if } j = k \\
  ch_{ij}(t), & \text{otherwise}
\end{cases} \]

4.1.2 **Boundary mutation**  
Let k = u[1, n], p=u(0, 1) and set  
\[ ch_{ij}(t + 1) = \begin{cases} 
  L_j, & \text{if } j = k, p < 0.5 \\
  U_j, & \text{if } j = k, p \geq 0.5 \\
  ch_{ij}(t), & \text{otherwise}
\end{cases} \]

4.1.3 **Non-uniform mutation**  
Let k = [1, n], p = u(0, 1) and set  
\[ ch_{ij}(t + 1) = \begin{cases} 
  ch_{ij}(t) + (L_j - ch_{ij}(t)) f, & \text{if } j = k, p < 0.5 \\
  ch_{ij}(t) + (U_j - ch_{ij}(t)) f, & \text{if } j = k, p \geq 0.5 \\
  ch_{ij}(t), & \text{otherwise}
\end{cases} \]

where \( f = u(0,1) \left( \frac{T_i}{T_0} \right)^\theta \)
and b is the shape parameter.

4.1.4 Multi-non-uniform mutation

Apply non-uniform mutation to all variables of CHi.

4.1.5 Simple cross-over

Let k= [1, n] and set
\[
ch_{i1}(t + 1) = \begin{cases} 
ch_{i1}(t), & \text{if } l < k \\
ch_{i2}(t), & \text{otherwise}
\end{cases}
\]

\[
ch_{j1}(t + 1) = \begin{cases} 
ch_{j1}(t), & \text{if } l < k \\
ch_{j2}(t), & \text{otherwise}
\end{cases}
\]

4.1.6 Simple arithmetic cross-over

Let k= u [1, n], p= u (0, 1) and set
\[
ch_{i1}(t + 1) = \begin{cases} 
pch_{i1}(t) + (1 - p)ch_{i2}(t), & \text{if } l = k \\
ch_{i2}(t), & \text{otherwise}
\end{cases}
\]

\[
ch_{j1}(t + 1) = \begin{cases} 
pch_{j1}(t) + (1 - p)ch_{j2}(t), & \text{if } l = k \\
ch_{j2}(t), & \text{otherwise}
\end{cases}
\]

4.1.7 Whole arithmetic cross-over

Simple arithmetic cross-over applied to all variables of CHi and CHj.

4.1.8 Heuristic cross-over

Let Scaling factor F = u (0, 1), j = [1, n] and k= [1, n] such that j ≠ k and set
\[
CH_{i1}(t + 1) = CH_{i1}(t) + F \cdot (CH_{j1}(t) - CH_{k1}(t))
\]

IMPLEMENTATION OF RASA ALGORITHM FOR SCHEDULING DEPENDENT TASKS

The following subsections deal with the representation of solution, and the generation of initial solution.

4.1 Solution representation

The solution is represented as an array of length equal to the number of jobs [8]. The value corresponding to each position i in the array represent the node to which task i was allocated. The representation of the solution for the problem of scheduling 13 tasks to 3 computing nodes is illustrated in Figure 1. The first element of the array denotes the first task (n1) in a batch which is allocated to the computing node 2; the second element of the array denotes the second job (n2) which is assigned to the computing node 1, and so on.
4.2 Initial solution generation
Numerous methods have been proposed to generate the initial solution when applying metaheuristics to the scheduling problem in the heterogeneous environment [9, 10]. Random solution may also be generated to initiate the process.

4.3 Computational experiments
To illustrate, a small scale DAG scheduling problem involving 3 nodes and 10 tasks is considered (Fig 2) with the computation cost matrix given in Table 1. The upward rank and the order of the tasks for execution are given in Table 2 and 3 respectively. RASA algorithm was executed with the following parameters. Size of the population -15, $T_{frac} -10^2$, $T_{frac\_min} -10^4$, $T_{mult}\_0.9$, Parameter $q -0.09$, Scaling factor -0.8, successmax(SM)-2, Number of iterations- 50. The makespan value obtained for the example problem is found to be 73.

Table 1  Computation cost matrix for random DAG

<table>
<thead>
<tr>
<th>Task id</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>14</td>
<td>16</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>13</td>
<td>19</td>
<td>18</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>13</td>
<td>19</td>
</tr>
<tr>
<td>4</td>
<td>13</td>
<td>8</td>
<td>17</td>
</tr>
<tr>
<td>5</td>
<td>12</td>
<td>13</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>13</td>
<td>16</td>
<td>9</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>15</td>
<td>11</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>11</td>
<td>14</td>
</tr>
<tr>
<td>9</td>
<td>18</td>
<td>12</td>
<td>20</td>
</tr>
<tr>
<td>10</td>
<td>21</td>
<td>7</td>
<td>16</td>
</tr>
</tbody>
</table>

Figure 2. An example DAG
Table 2: Upward rank of the tasks

<table>
<thead>
<tr>
<th>Task id</th>
<th>Upward Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>108.000000</td>
</tr>
<tr>
<td>2</td>
<td>77.000000</td>
</tr>
<tr>
<td>3</td>
<td>80.000000</td>
</tr>
<tr>
<td>4</td>
<td>80.000000</td>
</tr>
<tr>
<td>5</td>
<td>69.000000</td>
</tr>
<tr>
<td>6</td>
<td>63.33333</td>
</tr>
<tr>
<td>7</td>
<td>42.66667</td>
</tr>
<tr>
<td>8</td>
<td>35.66667</td>
</tr>
<tr>
<td>9</td>
<td>44.33333</td>
</tr>
<tr>
<td>10</td>
<td>14.66667</td>
</tr>
</tbody>
</table>

Table 3: Execution Order of tasks

<table>
<thead>
<tr>
<th>Order of Task id</th>
<th>Task id</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>9</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>10</td>
</tr>
</tbody>
</table>

Figure 3. A schedule produced by the RASA algorithm for the example DAG

CONCLUSION
The methodology of implementation of RASA algorithm for the constrained dependent task scheduling problem had been discussed. The methodology adopted for this algorithm gives rise to the development of scheduling algorithms using other meta heuristic methods.

REFERENCES


vi Mohammad I. Daouda, Nawwaf Kharma, A hybrid heuristic–genetic algorithm for 
task scheduling in heterogeneous processor networks, J. Parallel Distrib. Comput. 71 
(2011) 1518–1531

vii Hrstka, O., Kučerová, A., Lepš, M., Zeman, J.: A competitive comparison of different 

viii Selvi, S, Manimegalai, D. Task Scheduling using Two Phase Variable Neighborhood 
Search Algorithm on heterogeneous computing and grid environments, Arabian 

ix Abraham A, Liu H, Zhao M. Particle swarm scheduling for work-flow applications in 
distributed computing environments. Studies in Computational Intelligence 2008; 
128: 327–342.

x Xhafa F, Duran B, Parallel memetic algorithms for independent job scheduling in 
computational grids, in: Recent Advances in Evolutionary Computation for 
Combinatorial Optimization, vol. 153 of Studies in Computational Intelligence, 