CONTENTS

1. System Model

What is a System Model ?

A system model represents the aspects of a system and its Environment.

System modelling means representing the detailed view of the system using some kind of Objectives.

Listed below are some of the Objectives which I considered in the Proposed Algorithm.

Objectives :

1. Makespan
2. Energy Consumption
3. Load balancing
4. VM Utilization
5. Speedup
6. SLR

1. Makespan :

The makespan (MS) is the overall processing time of a workflow application.In the Proposed Algorithm it is Given that '' the makespan of a given DAG can be calculated as the time at which the execution of the last task, T_{exit}, would be complete''.

$$\mathbf{MS= AF\ T\ (T_{exit})}$$

MS = makespan

AFT = Actual finish time of task T_i on Vm_j

Here Actual finish time of task T_i on Vm_j ( ) can be calculated using the earliest start time ($EST_{i,j}$) of a task and Execution time ($ET_{i,j}$) of the Task .

$$\mathbf{AFT_{i,j} = EST_{i,j} + ET_{i,j}}$$

Here $\mathbf{EST}(T_{i,j}) = \max_{Ti \in pred(Tj)} \{AF\ T(T_i)\}$

for Entry tasks EST is zero i.e., $\mathbf{EST_{entry} = 0}$

and the execution time of given tasks, we represent the computational speed of a virtual machine VM_j as SV_{Mj} and performance variability is represented as D. Weight of each node WTi represents the total number of instructions in MI, required to finish a given task Ti. Thus, the time taken to execute task Ti on VMj considering decrement in voltage supply by dVi and the relative decrement in VM speed as SP_{dVi} , is given as follows :

$$ET_{i,j} = \frac{W_{T_i}}{S_{VM_j} \times \left(1 - D - SP_{dv_i}\right)}$$

In the proposed algorithm speed of a virtual machine VM_j as SV_{Mj} is given as

VM-ID	VM speed (in MIPS)
VM_1	4
VM_2	2.5
VM_3	2

and

D=0.24 (taken from Amazon EC2)

2. Energy Consumption :

In the proposed algorithm the energy consumed during the execution of a particular workflow is defined as follows :

$$Ec = 2 \times \beta \times M \times IV \times \sum_{i=1}^{n} dV_i$$

Here

Ec = Energy conserved

β = constant (0.66 as i taken in proposed algorithm)

M= makespan

IV = Input Voltage (230 volts as standard)

dV_i = Is the decrement applied in voltage of the VM on which task T_i is running

3. Load balancing :

Load balancing factor is used to check whether the load distributed among all VMs is equal or not . Load balancing is 100% and that means load distributed among all VMs are equal.
Load is measured only based on the Active Time (AT) . If all tasks are almost equally distributed among 3 available VMs, variance among the all VMs will be less. To measure variance among all VMs we need to calculate standard deviation (SD) of AT of VMs.

It can be calculated as follows :

$$AT(VM_j) = \sum_{i=1}^{x} EET(i, j) * \delta(i, j)$$

Where,

$$\delta(i, j) = \begin{cases} 1, & \text{if } t_i \text{ is assigned to } VM_j \\ 0, & \text{otherwise} \end{cases}$$

The average value of AT of all the VMs, it can be calculated as follows

$$Avg(AT) = \frac{\sum_{j=1}^{y} AT(VM_j)}{y}$$

$$SD = \sqrt{\frac{\sum_{j=1}^{y} (AT_j - Avg(AT))^2}{y}}$$

AT_j = the active time of the VM_j

SD =standard deviation

y= no.of VM's

EET = Expected Execution Time (or Execution time ($ET_{i,j}$) of the Task)

If SD is minimized then load balance is maximized. If the SD of load on all VMs is zero then load balancing is 100% and that means load distributed among all VMs are equal.

4. VM Utilization :

It is the ratio of the active time of the VMs to the MKS. Mathematically represented as follows :

$$Utlization(VM_j) = \frac{AT(VM_j)}{MKS}, Where\ j\ =\ 1\ to\ y$$

AT_j = the active time of the VM_j

MKS = makespan

y = no.of VM's

Average VM utilization :

Average VM utilization is the utilization of all the VMs in the cloud data center. Mathematically represented as follows :

$$Avg\ U(VM_j) = \frac{1}{y}\sum_{i=1}^{y} Utlization(VM_j)$$

5. Speedup :

It is the ratio of maximum sequential execution of VMs by parallel execution schedule length. It can be mathematically represented as follows :

$$Speedup = \frac{\sum_{i=0}^{x} EET(T_i, V_j)}{MKS}$$

MKS = makespan

EET = Expected Execution Time (or Execution time ($ET_{i,j}$) of the Task)

6.Schedule Length Ratio(SLR) :

To measure the performance of any scheduling algorithm on the DAG is the schedule length (MKS) of its output schedule. Because a different size of graphs are used, it is required to normalize the MKS to lower bound, which is called SLR. SLR mathematically represented as follows :

$$SLR = \frac{MKS}{\sum_{n_i \in CP_{MIN}} min_{P_j \in Q} W_{(i,j)}}$$

MKS = makespan

CP_{min} = minimum computation costs ,(For an unscheduled DAG, if the computation cost of each node n_i is set with the minimum value, then the critical path will be based on minimum value, then the critical path will be based on the minimum computation costs, which is represented as CP_{min})

The denominator is the summation of the minimum computation costs of tasks on the CP_{min}

2. Proposed Algorithm

Proposed Algorithm :An Energy Efficient Algorithm for Workflow Scheduling in IaaS Cloud (EEWS) .

Abstract:

Energy efficient workflow scheduling is the demand of the present time's computing platforms such as an infrastructure-as-a-service (IaaS) cloud. An appreciable amount of energy can be saved if a dynamic voltage scaling (DVS) enabled environment is considered. But it is important to decrease the makespan of a schedule as well, so that it may not extend beyond the deadline specified by the cloud user .This Algorithm is inspired from hybrid chemical reaction optimization (HCRO) algorithm.

Objectives of Algorithm :

Objective 1: Minimize $M = AFT(T_{exit})$

Objective 2: Maximize $Ec = 2 \times \beta \times M \times$,

$$\times \sum_{i=1}^{n} dV_i$$

Detailed Explanation of The Problem Formulation and Terminologies of algorithm :

Let us Consider a workflow is modeled as a DAG, $W_f = (T , E)$, where $T = \{T_1, T_2, \cdots , Tn\}$ is a set of n tasks and $E = \{E_1, E_2, \cdots , Ee\}$ represents a set of e edges. The weight assigned to a node indicates the number of instructions in MI (millions of instructions) required to finish a given task.An edge $E_k = (T_i, T_j)$ indicates the precedence relationship between task T_i and task T_j . In other words, if an edge $E_k = (T_i, T_j)$ exists in set E, then task Ti is a predecessor to task T_j and hence the execution of task T_j can begin only after task T_i is complete.

Cloud providers offer a variety of heterogeneous VMs with different processing capabilities.

In the proposed algorithm, the cloud model consists of a set of m heterogeneous VMs denoted by R = {VM_1,VM_2 ⋯ ,VM_m}. The processors of the physical machines on which VMs are deployed are DVS-enabled, i.e., they can work on different input voltages but any decrease in voltage supply causes a relative decrement in processor speed as shown in Table below :

Decrement in voltage (dV) in volts	Relative decrement in speed (sp_{dV})
0	0
0.25	0.05
0.5	0.125
1.0	0.175
1.25	0.25

Note : dV << IV i.e the amount of decrement applied in the input voltage is relatively much lesser than the standard value of the input voltage itself.

Reason Why dV << IV :

This is because the machines have a small range of workable voltages and if the input voltage drops below this range, a physical machine may not work. It is also assumed that the tasks of a particular workflow are scheduled at the same data center and the transfer time required for transmitting data from one VM to another VM is negligible with respect to the task computation time. So, the data transfer time does not have a huge impact on the makespan, and hence it can be ignored.

Notations and definitions :

Notations	Definitions
Pop	Current Population
$PopSize$	Number of molecules in the Pop
V_i	i^{th} molecule
V_{best}	$best$ molecule
T_j	j^{th} task (an atom in V_i)
VM_k	k^{th} Virtual Machine
$PE(V_i)$	Potential Energy of molecule V_i
$KE(V_i)$	Kinetic Energy of molecule V_i
M	Makespan
M_n	Normalized value of M
M_{max}	Maximum makespan
Ec	Energy conserved
$Econ$	Energy consumed
Ec_n	Normalized value of Ec
Ec_{max}	Maximum energy conserved
IV	Standard input voltage
W_{T_i}	Length of task T_i
T_{exit}	Last task of the DAG
$fitness$	fitness value
D	processor performance variability on scale of 0-1
$ET_{T_j}^{VM_k}$	Execution time of task T_j on VM_k
sp_{dV_j}	Decrement in speed of processor on scale of 0-1 where T_i is running
dV_j	Reduction in the input voltage of processor where T_i is running
$type$	parameter that determines whether to perform a single molecule or an intermolecular operator
$intertype$	parameter that determines whether to perform an intermolecular ineffective collision or synthesis

Fitness Function:

As stated earlier, the physical machines are considered to be DVS-enabled. We formulate the fitness function for our problem as follows :

Potential Energy (P E) is a measure of the goodness of a molecule (lower the P E, better is the corresponding schedule). So, the potential energy function should be constructed in such a way that it evaluates the given molecule accurately. A potential energy function may be built by using one or more than one objective. In our case, it comprises two objectives . In our case, it comprises two objectives, with makespan minimization as primary and maximization of energy conservation as our secondary objective.

They are :

Objective 1: Minimize $M = AFT(T_{exit})$

Objective 2: Maximize $Ec = 2 \times \beta \times M \times$

$$\times \sum_{i=1}^{n} dV_i$$

Here Fitness function is the PE (potential energy of a molecule) and it is calculated as follows :

$$PE = \gamma \times M_n + \frac{\delta}{(1 + Ec_n)}$$

The Molecule which is having minimum PE after all desired no of iterations is considered as the Best Molecule, here minimized PE is Given as follows

$$\text{Minimize } PE = \gamma \times M_n + \frac{\delta}{(1 + Ec_n)}$$

subject to the following constraints:

$M < M_{max}$

$0 < Ec < Ec_{max}$

$\gamma = 1 - \delta$ and $0 < \gamma < 1$

Generation Population:

Here number of possible schedules of workflow are **n!** Among n! numbers of schedules I took some random schedules based on Population size as my population for each iteration.

Generation Of Molecules Of Initial Population :

To generate the initial population ,A molecule is selected at random from the population. Then, an atom (A task in a molecule is called an atom.)is chosen from the selected molecule and its last predecessor and first successor are determined. After this, a left (or right) rotation is performed, starting from the chosen atom till the atom which is just after its predecessor (or just before its successor), in order to satisfy the precedence constraints.

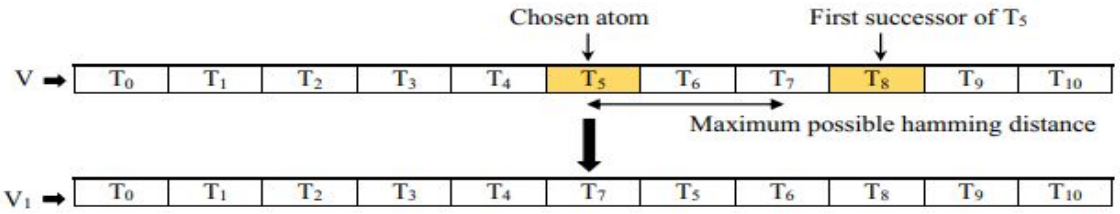


Fig .An illustration of generation of molecules of the initial population

This process is repeated until the required number of molecules is obtained.

Five operators we need to calculate in order to implement proposed algorithm

They are listed below :

- **On-Wall Ineffective Collision**
- **Decomposition**
- **Intermolecular Ineffective Collision**
- **Synthesis**
- **on-wall pseudo-effective collision**

On-Wall Ineffective Collision :

A single molecule is chosen as a reactant. To derive a child molecule V1 from parent molecule V a right rotation is performed from the chosen atom towards its last predecessor. Note that, for this operation the condition P E(V1) ≤ P E(V) + KE(V) must hold true.

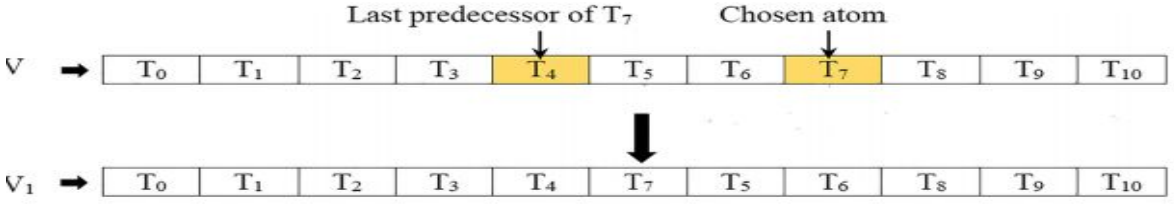


Fig.An example of on-wall ineffective collision

Decomposition :

This operator involves disintegration of a single molecule V into two new molecules V1 and V2. An atom is chosen from the given molecule and a left or right rotation is performed either towards its first successor or towards its last predecessor. But the condition P E(V1) + P E(V2) ≤ P E(V) + KE(V) + buf energy (buf energy is energy stored in the shared buffer) should hold true.

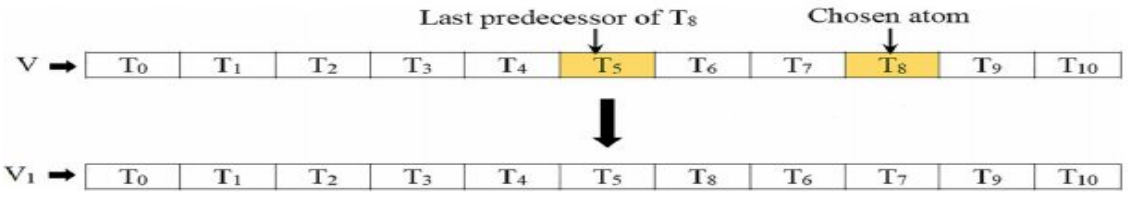


fig.An example of decomposition

Intermolecular Ineffective Collision :

This reaction involves collision of two molecules, V1 and V2, resulting in the formation of two new molecules, V 1 and V 2 using crossover operation. But the condition, P E(V 1) + P E(V 2) ≤ P E(V1) + P E(V2) + KE(V1) + KE(V2) should hold. Then, V 1 and V 2 are compared and the one with higher potential energy is rejected.

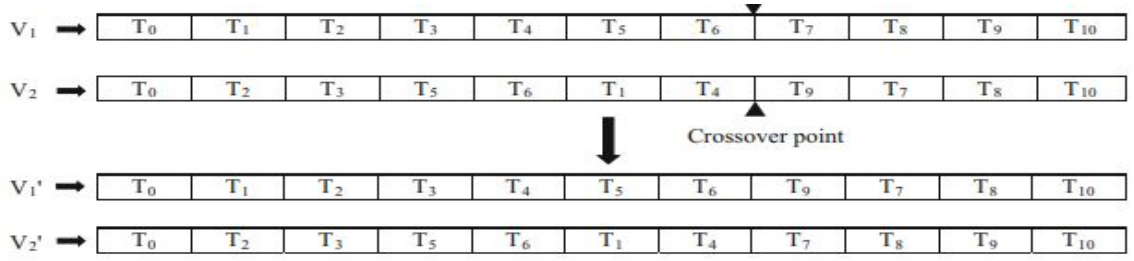


Fig. An example of intermolecular ineffective collision

Synthesis :

Synthesis is an intermolecular reaction in which two molecules fuse to form a single molecule, using crossover operation. In synthesis, two molecules V1 and V2 collide to

produce two new molecules V 1 and V 2 such that P E(V 1) ≤ P E(V1) + P E(V2) + KE(V1) + KE(V2) and P E(V 2) ≤ P E(V1)+P E(V2)+KE(V1)+KE(V2) and then the molecule with lower potential energy is chosen as the resultant molecule V .

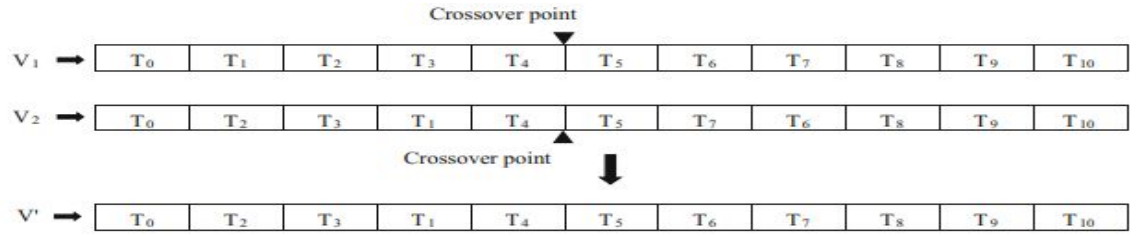


Fig. An example of synthesis

Proposed Algorithm :

Algorithm 1 Energy efficient workflow scheduling.

1: **procedure** EEWS($W_f(T, E)$)
2: **Input:** Given DAG $W_f(T, E)$, $type$, $intertype$,KE_{Intial}, $KE_{LossRate}$
3: **Output:** Best molecule V_{best}
4: Initialize Pop
5: Calculate PE of each molecule in Pop using (15) after mapping as per (20)
6: **while** $iteration \neq max\_num\_iterations$ **do**
7: **for** $l = 1$ to $PopSize$ **do**
8: Select some molecules and form a set $Parent$
9: **end for**
10: **for each** molecule V_i in $Parent$ **do**
11: Choose a random number r_1
12: **if** $r_1 \leq type$ **then**
13: V_{i_1} = Perform on-wall ineffective collision on V_i
14: V_{i_1} = Perform decomposition V_{i1}
15: **else**
16: Select another molecule V_k from $Parent$
17: Choose a random number r_2
18: **if** $r_2 < intertype$ **then**
19: V_{i_1} = Perform intermolecular ineffective collision between V_i and V_k
20: **else**
21: V_{i_1} = Perform synthesis between V_i and V_k
22: **end if**
23: **end if**
24: $V_{i2} = Call\ ON-WALL-PSEUDO-EFFECTIVE(V_{i1})$
25: Calculate PE of V_{i1} and V_{i2}in Pop using (15) after mapping as per (20)
26: **if** $PE(V_{i1}) \leq PE(V_{i2})$ **then**
27: $V_i = V_{i1}$
28: **else**
29: $V_i = V_{i2}$
30: **end if**
31: $Pop = Call\ REPLACE(Pop, PopSize, V_i)$
32: **end for**
33: **end while**
34: **return** V_{best}
35: **end procedure**

Explanation :

In line 2 input is taken as DAG matrix and its Weight of each node given and type,intertype,$KE_{initial}$,$KE_{lossRate}$ values are listed below :

Parameter	Value
$PopSize$	500
$KE_{initial}$	0.1
$KE_{LossRate}$	10%
$type$	0.4
$intertype$	0.5

In lines 4-5, we initialize the initial molecule population and calculate the potential energy of each molecule After as per the given below mapping function :

$$Map(T_i, VM_k) = \underset{VM_k \in avail\ VMs}{Min} \{ET_{T_i}^{VM_k}\}$$

here i taken

D=0.24

PopSize=4

Then, we start the process of finding a near optimal solution as given in lines 6 through 28 and repeat the same, until we reach the maximum specified number of iterations. In lines 7-9, we select some molecules randomly from the current population, and form a Parent set. Followed by this, we generate a random number r1 in line 11. Based on the value of r1, we determine the operator which will be used in that iteration. As given in line 12, if the value of r1 is less than the value of type, we use two single-molecule operators successively on the given molecule Vi. Otherwise, we use an intermolecular operator as given in lines 18 through 22 (for which we find another reactant molecule Vk as in line 16) based on the comparison between the values of another random number r2 and the input parameter intertype. In line 24, we use a novel operator called on-wall pseudo-effective collision on Vi1 to form Vi2, .In line 25, we calculate the potential energy of the newly generated molecules Vi1 and Vi2. Next, we compare the potential energy of Vi1 and Vi2 and run Algorithm 3 (as explained later) on the one with a lesser value.

Algorithm 2 On-wall pseudo-effective collision.

1: **procedure** ON-WALL PSEUDO-EFFECTIVE(V)
2: **Input:** A molecule V.
3: **Output:** A new molecule V_{new}.
4: Let $V_{new} \leftarrow V$
5: Choose randomly an atom T_i from V_{new}
6: Find the last predecessorT_j of the atom T_i
7: Find the first successor T_k of the atom T_i
8: Find a random number $rpos$ where
$p_{T_{j+1}} \leq rpos \leq p_{T_{k-1}}$ such that $rpos \neq p_{T_i}$
9: Let $T' \leftarrow$ atom at $rpos$ in V_{new}
10: Exchange $T_i \leftrightarrow T'$ in V_{new}
11: **return** V_{new}
12: **end procedure**

Explanation :

According to the algorithm, we randomly select an atom Ti from the given molecule and find its last predecessor Tj and first successor Tk (as given in lines 5 through 7). Let the position of atoms Ti, Tj , Tk in the molecule be denoted by p_{Ti} , p_{Tj} , p_{Tk} respectively. In line 8, a random number, say rpos, is generated.between $p_{T\,j}$ and p_{Tk} , provided rpos is not equal to pTi. This furnishes the chosen atom with the maximum number of alternatives for swapping. Then the atom present at rpos (T) is swapped with Ti to obtain a new molecule Vnew, as given in lines 9-10. Whether the molecule Vnew enters the current population or not, depends on the criteria mentioned in the Algorithms 1 and 3.

Algorithm 3 Replacing the current molecules.

```
1: procedure REPLACE(Pop, PopSize, V_new)
2: Input: Current population Pop, A new molecule
   V_new
3: Output: Modified Pop.
4:     V_worst ← V_1
5:     V_best ← V_1
6:     for i = 1 to PopSize do
7:         if PE(V_i) ≤ PE(V_best) then
8:             V_best = V_i
9:         else if PE(V_i) ≥ PE(V_worst) then
10:            V_worst = V_i
11:        end if
12:    end for
13:    if PE(V_new) ≤ PE(V_best) then
14:        replace V_best by V_new in Pop
15:        replace V_best by V_new in buffer
16:    else if PE(V_new) ≤ PE(V_best) + KE(V_best)
   then
17:        replace V_best by V_new in Pop
18:    else if PE(V_new) ≤ PE(V_worst)+KE(V_worst)
   then
19:        replace V_worst by V_new in Pop
20:    end if
21:    Return Pop
22: end procedure
```

Explanation :

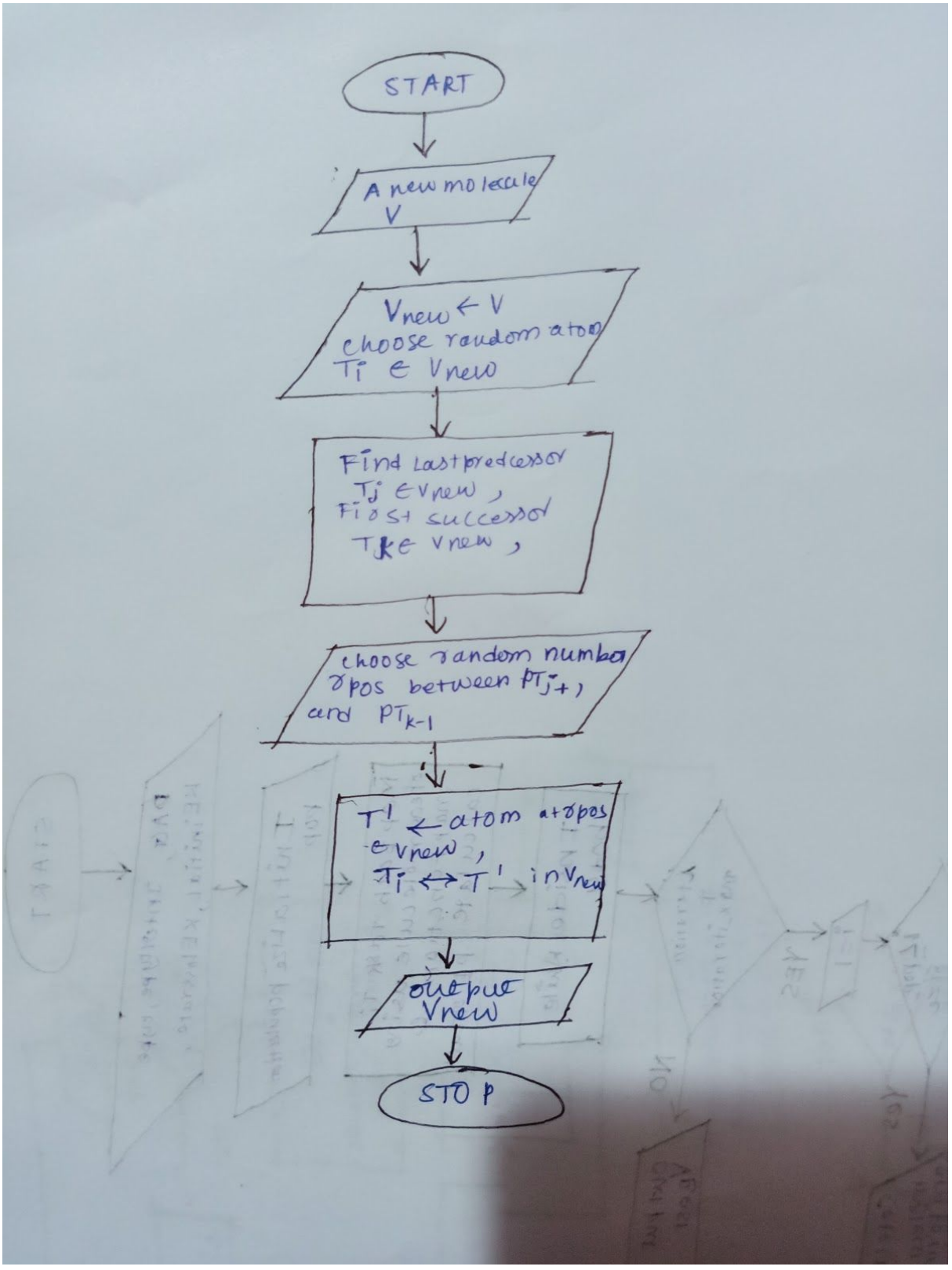
Here we propose a new scheme of including molecules in the current population, as given in Algorithm 3. The decision whether the molecule formed by applying any of the five operators should be allowed to enter the current population, is dependent on an entirely different set of conditions than those in HCRO. According to the algorithm, a new molecule (Vnew) produced by using any of the five operators, is considered as a potential candidate for being included in the current population. To determine its eligibility, we find the best (Vbest) and the worst (Vworst) molecule in the current population based on the measure of their potential energy as given in lines 6 through 12. In lines 13-15, we check whether the best molecule can be replaced in the population as well as in the buffer (which stores the best-so-far molecule in terms of potential energy, so that it may not be replaced in the current population). If not so, in lines 16-17, we verify whether we can replace the best molecule at least in the population (without any change in the buffer).If this too does not take place, the new molecule is compared to the worst, as in lines 18-19. If even this condition is false, we do not include the new molecule in the population. As mentioned in previous sections, KE is used to increase

tolerance of the algorithm towards those molecules which do not appear worthy at present, but may become important later due to some minute modifications performed by the five aforementioned operators.

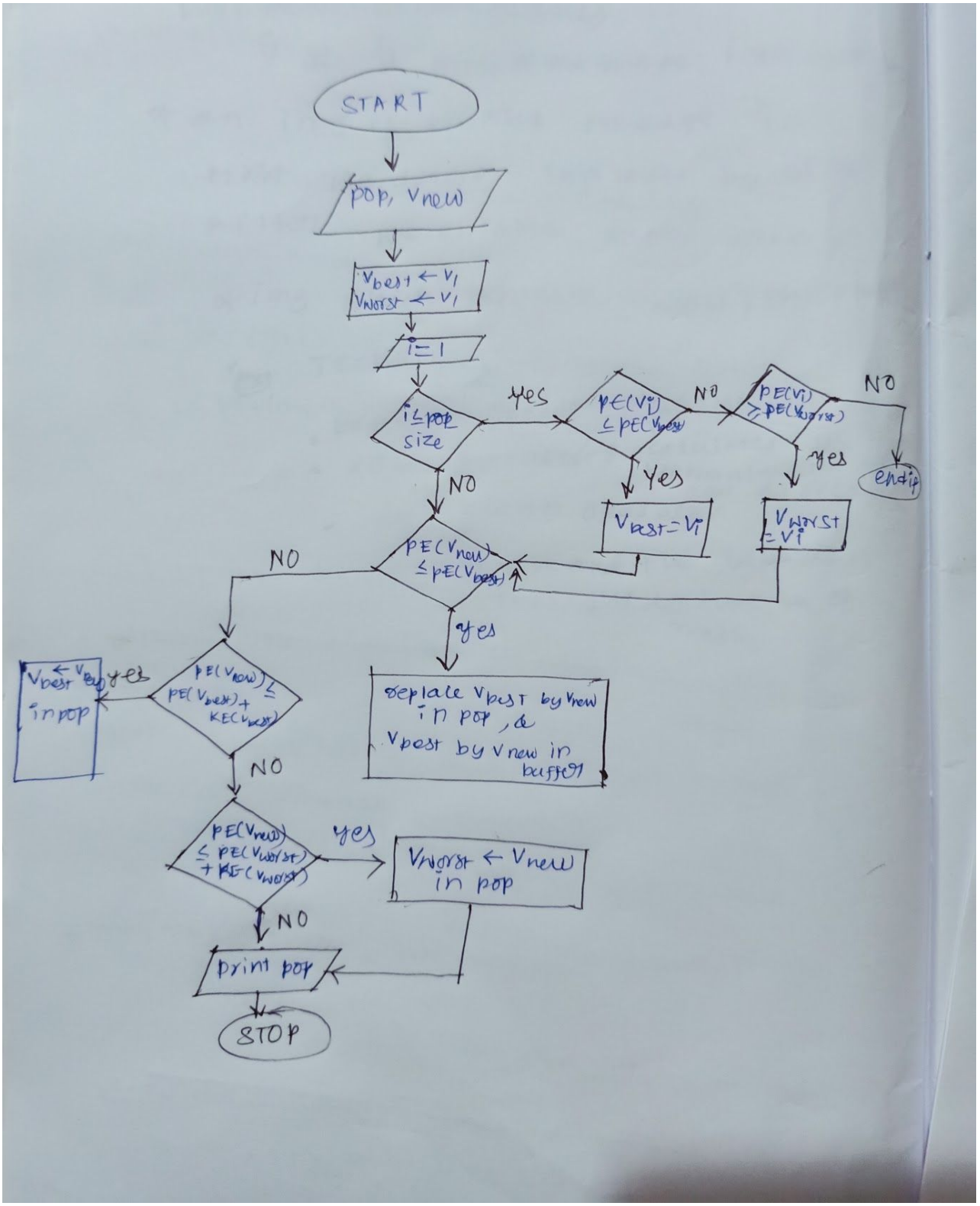
3. Flowchart Of The Algorithm

Flowchart of Algorithm :

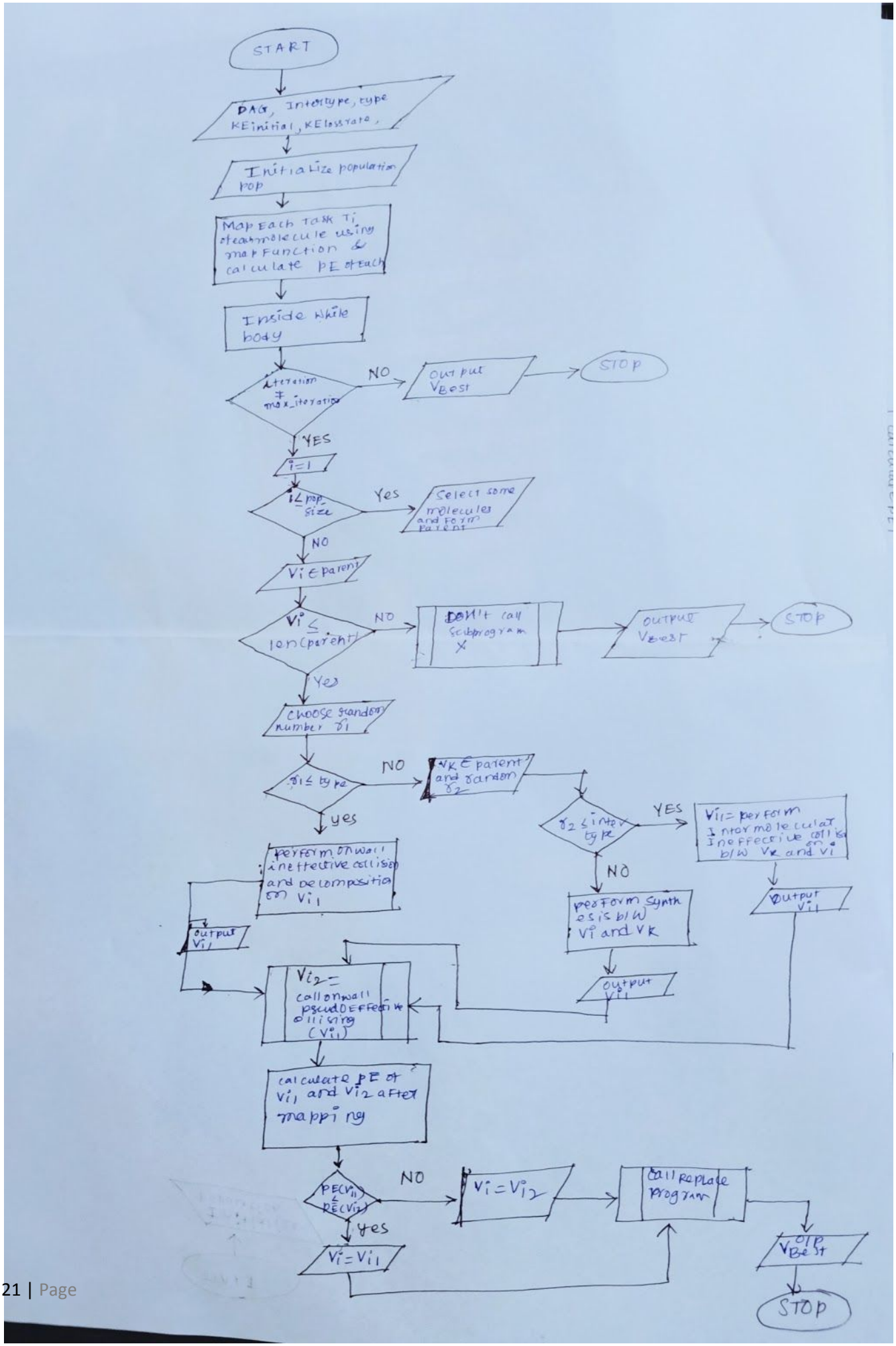
Algorithm-2:


START
A new molecule V
Vnew ← V
choose random atom Ti ∈ Vnew
Find Last predecessor Tj ∈ Vnew , First successor Tk ∈ Vnew ,
choose random number γpos between PTj+1 and PTk-1
T' ← atom at γpos ∈ Vnew , Ti ↔ T' in Vnew
output Vnew
STOP

Algorithm-3 :



Algorithm -1 :


START
DAG, Intertype, type
KEinitial, KElossrate,
Initialize population pop
Map Each task Ti of eachmolecule using map Function & calculate PE of Each
Inside While body
Iteration ≠ max_iteration
NO
Output VBest
STOP
YES
i=1
i<pop_size
Yes
Select some molecules and Form Parent
NO
Vi ∈ parent
Vi ≤ len(parent)
NO
Call subprogram X
output VBest
STOP
Yes
Choose random number γ1
γ1 ≤ type
NO
Vk ∈ parent and random γ2
yes
γ2 ≤ inter type
YES
Vi1 = perform intermolecular Ineffective coll bw Vk and Vi
output Vi1
NO
perform synthesis b/w Vi and Vk
output Vi1
perform onwall ineffective collision and decomposition on Vi1
output Vi1
Vi2 = callonwall pseudoEffective colliding (Vi1)
calculate PE of Vi1 and Vi2 after mapping
PE(Vi1) ≤ PE(Vi2)
NO
Vi = Vi2
Call Replace program
yes
Vi = Vi1
OTP VBest
STOP

4. Time Complexity Of Algorithm

The the overall time complexity of Algorithm 1 is O(max num iterations × P opSize × (n2 + n × e + P opSize))

The overall time complexity of Algorithm 2 is O(n × e)

where e = the number of edges

n =the total number of tasks in the given workflow

Algorithm 3, the time complexity is O(PopSize),

5.Example

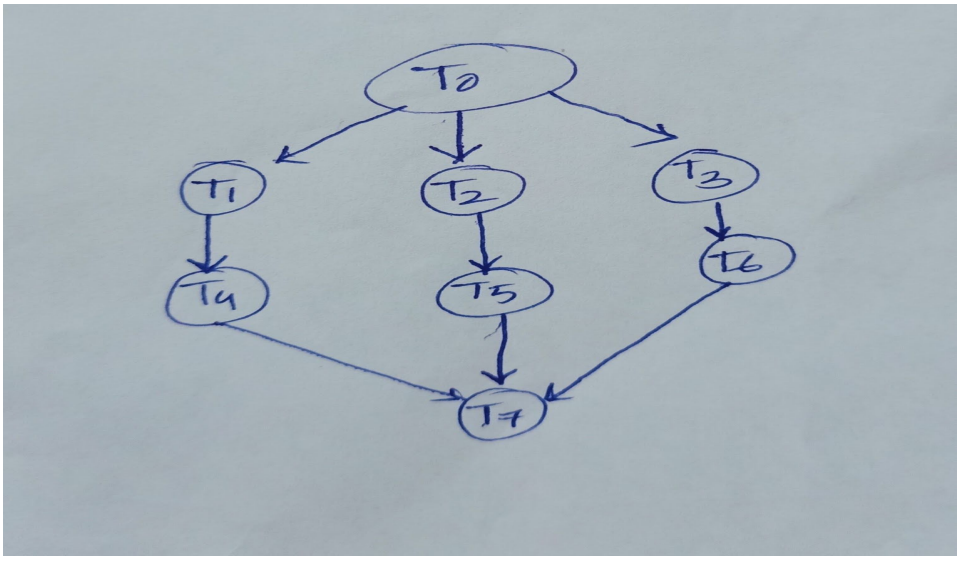
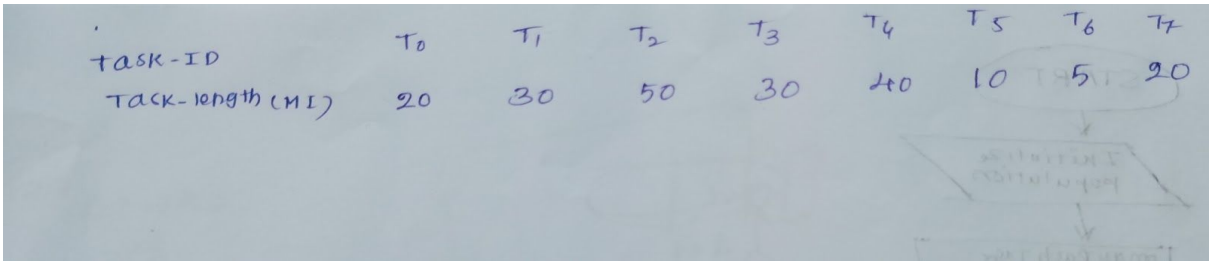


Fig.An example of a workflow

Table -1 :Length of tasks in the given DAG

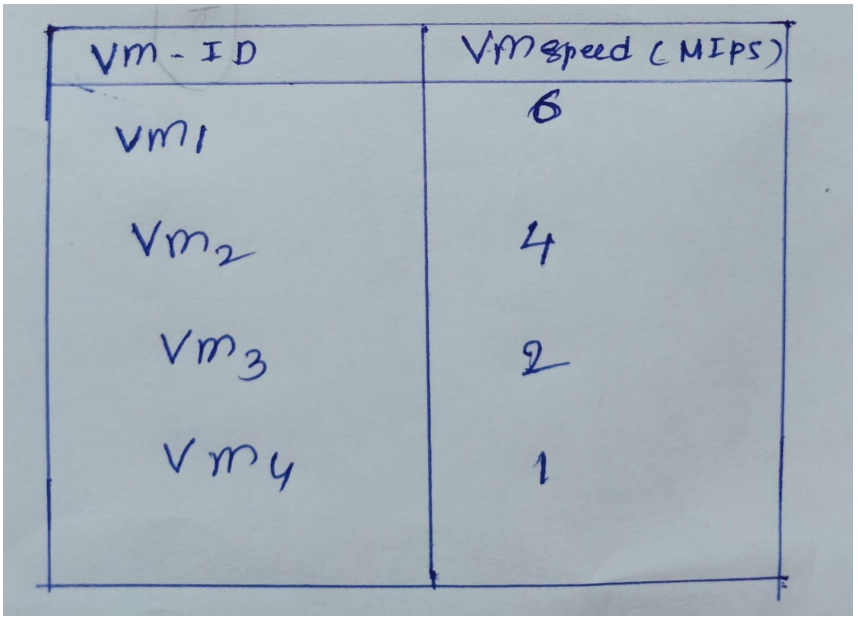


task-ID	T_0	T_1	T_2	T_3	T_4	T_5	T_6	T_7
Task-length (MI)	20	30	50	30	40	10	5	20

Table -2 : Effect of reduction of voltage on processor speed

Decrement in voltage (dV) in volts	Relative decrement in speed (sp_{dV})
0	0
0.25	0.05
0.5	0.125
1.0	0.175
1.25	0.25

Table -3 :VM information



VM - ID	VM speed (MIPS)
VM_1	6
VM_2	4
VM_3	2
VM_4	1

Initial population :

Molecule TaskID

Molecule 0 [0, 1, 4, 3, 2, 5, 6, 7]

Molecule 1 [0, 1, 2, 3, 4, 6, 5, 7]

Molecule 2 [0, 1, 2, 3, 4, 5, 6, 7]

Molecule 3 [0, 1, 2, 3, 4, 6, 5, 7]

PE of Molecule_ 0 in Modified 0.40299170616113744

PE of Molecule_ 1 in Modified 0.3851190476190476

PE of Molecule_ 2 in Modified 0.3857356395472987

PE of Molecule_ 3 in Modified 0.3851190476190476

After First Iteration :

Molecule 0 [0, 1, 4, 3, 2, 5, 6, 7]

Molecule 1 [0, 1, 2, 3, 6, 5, 4, 7]

Molecule 2 [0, 1, 2, 3, 4, 6, 5, 7]

Molecule 3 [0, 3, 2, 1, 4, 5, 6, 7]

PE of Molecule_ 0 in Modified 0.40469625161568296

PE of Molecule_ 1 in Modified 0.40288979298791433

PE of Molecule_ 2 in Modified 0.3867307692307692

PE of Molecule_ 3 in Modified 0.38734736115902035

After last iteration :

Molecule 0 [0, 1, 2, 3, 4, 6, 5, 7]

Molecule 1 [0, 1, 4, 3, 2, 5, 6, 7]

Molecule 2 [0, 1, 2, 3, 6, 5, 4, 7]

Molecule 3 [0, 1, 2, 7, 4, 3, 6, 5]

PE of Molecule_ 0 in Modified 0.3867307692307692

PE of Molecule_ 1 in Modified 0.40469625161568296

PE of Molecule_ 2 in Modified 0.40288979298791433

PE of Molecule_ 3 in Modified 0.38987003863716196

Final the best molecule after all iterations : [0, 1, 2, 3, 4, 6, 5, 7]

Its fitness value is : 0.3851190476190476

makespan of Best molecule : 24.12280701754386

Energy conserved of Best Molecule : 21971.05263157895