CHAPTER-3
OPTIMIZATION METHODS USED

This chapter contains all the Optimization methods used containing algorithms, their technological descriptions, pseudo codes for existing work and Proposed method. The existing algorithms like Greedy Nearest Neighbor, Simulated Annealing, Ant Colony Optimization algorithm and Genetic Algorithms have been studied. Based on the study, it is commended for the design and development of the proposed method referred as an Adaptive Dynamic Genetic Algorithm.

3.1 Greedy Nearest Neighbor Algorithm

Usually, algorithms designed for optimization problems normally follow series of steps, with some set of choices at each step. The greedy algorithm always selects the choice that seems to be the best at that moment. That is, it selects a local optimum value with a hope that the selected choice will lead to a global optimum solution. However, the Greedy algorithm does not always give optimal solution, but for many problems it will give the solution [111]. The Greedy Algorithm takes decision incrementally in small steps without going back to the same state. The decision on each step is to improve the current state in a myopic (narrow minded) fashion without thinking of global situation. The decision could be fixed and often based on some simple priority rules. Here, the node selection is based on the priority that a node is an ideal one and not blocked for malicious data transfer. Let N={1,2….n} be the set of nodes available for use and each node is selected for task execution with a starting time of s_i and finishing time f_i, i.e, between $\{s_i, f_i\}$. Here, the execution times of all the tasks are sorted from least to the maximum. Then the task allocation to the nodes is selected greedily going down from the list by picking up which node is suitable for the current task. The running time depends on the type of sorting algorithm chosen for implementation. The sorting part can be as small as *O(n log n)* and the other part is *O(n)*.

Pseudo-code of Greedy Nearest Neighbor Algorithm

Rset is the set of all requests

Xset is empty (Xset will store all the tasks that will be scheduled)

While Rset is not empty

Choose i∈Rset such that finishing time of i is least

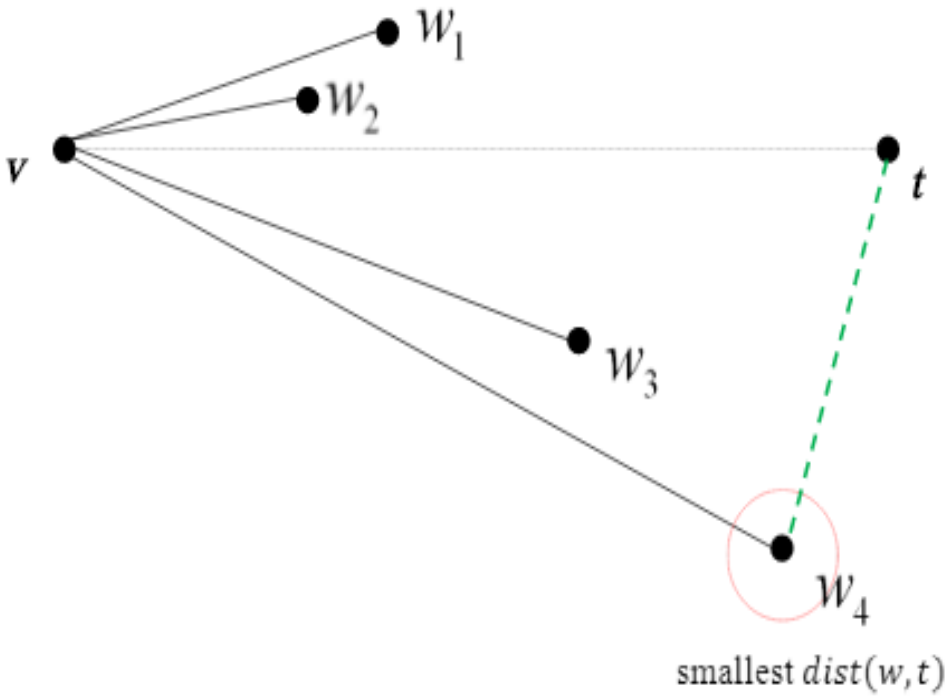
Add i to Xset

Remove from Rset all requests that overlap with i

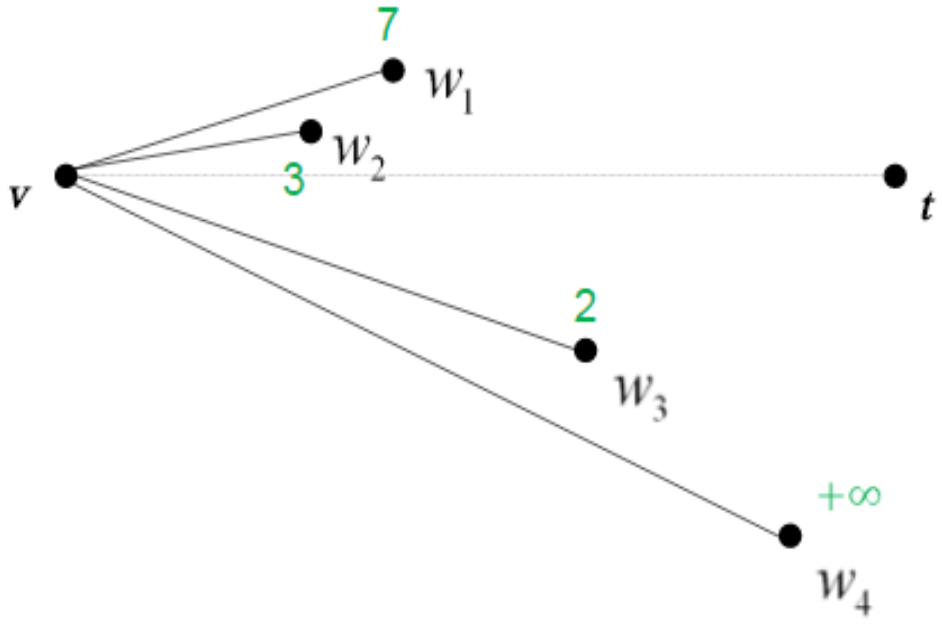
Return Xset

Considering the greedy nearest neighbor heuristic the sorting part can rise to $O(n^2)$. The nearest neighbor algorithm was introduced by J G Skellam and the work was continued by Clark and Evans. This procedure is followed to solve complex problems like travelling salesman problem [112]. The advantage of this algorithm is easy to design and implement, also runs fast since it is simple. Because of its nature of greediness, nearest neighbor algorithm misses best of solution that can be easily detected with insight. It is possible that nearest neighbor may not find a feasible solution even though there is one present in the arena[113]. The Algorithm works on arriving at global optima by selecting a local optimum nearby. The optimal solution to the problem is derived from optimal solution of the sub problem, looks like a dynamic programming. The worst case here is that it has to visit to all the nodes for selecting or choosing the best possible solution. Even then we cannot expect to obtain the overall optimum solution at the end.

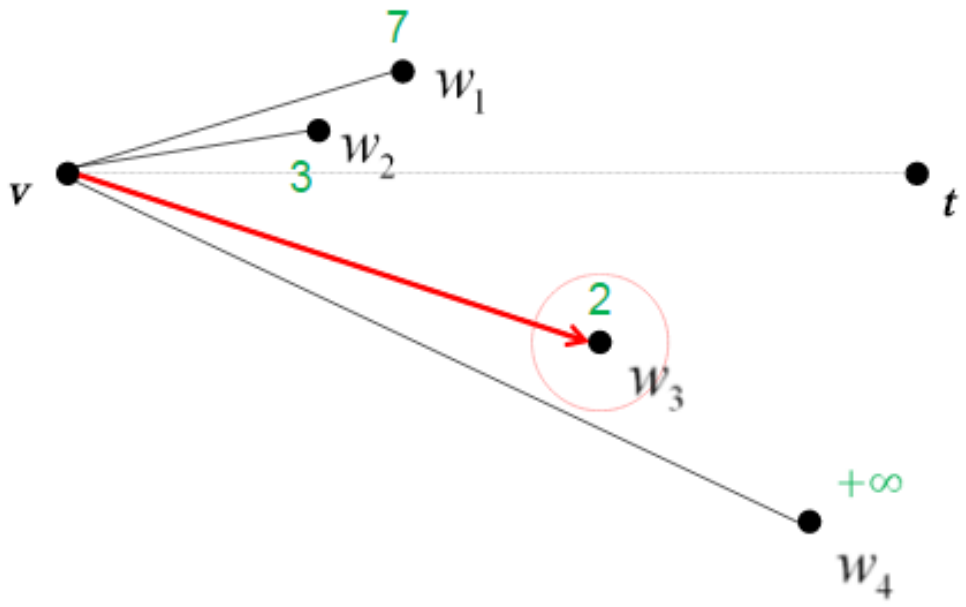
The Greedy routing behavior is illustrated in figure 3.1. The task allocation has to be done from current node which is represented by ‘v’ to destination node ‘d’. The greedy routing behavior is such that it will not jump directly to destination. It searches for the nearest neighbor node ‘w’ as the node nearest to ‘t’, a node in the direction of ‘d’, which are represented as w1, w2, w3 and w4 in the figure for easy representation.



(a)



(b)



(c)

Figure 3.1. (a) –(c) Greedy Routing Behavioral Diagrams

(Image Courtesy : Greedy Routing v t Finn, 1987)

The performance is based on objective function d={(w,v,t)}, where 'w' is the next node, 'v' is the current node and 't' is the terminal node.

F:d->zU{+∞}

The generalized greedy routing selects a neighbor w of v that minimizes f(w,v,t) in each iteration. The next node for 'v' is selected based on lowest weight value. In figure 3.1.(c) the 'w3' node is selected having lowest weigh value of '2'. The algorithm performs the next task selection to its neighbor nodes considering only local information around the current node, no information of the entire network is required.

3.2 Simulated Annealing

Simulated annealing is a meta-heuristic method which relies on local search [114]. Simulated annealing approach requires a scheduled representation as well as a neighbourhood operator for moving from the current solution to a required candidate solution. Annealing method allows the process to jump to worse solutions and thus often avoid local sub-optimal solution. Aarts, Laarhoven, and Lenstra described one

of the first simulated annealing processes for implementation to scheduling problems [115]. Simulated annealing is a metallurgical process of heating and then cooling it to a steady organized state.

The simulated annealing process is illustrated in figure 3.2 below. In this process the metal is heated to a molten state and cooled down to a molding state. If the temperature is reduced at a faster rate the heated atoms combine to form a polycrystalline structure which is more brittle in nature. Hence the temperature is reduced at slower and controlled rate retaining the minimum possible energy in bonding the atoms. The process of cooling is called annealing.

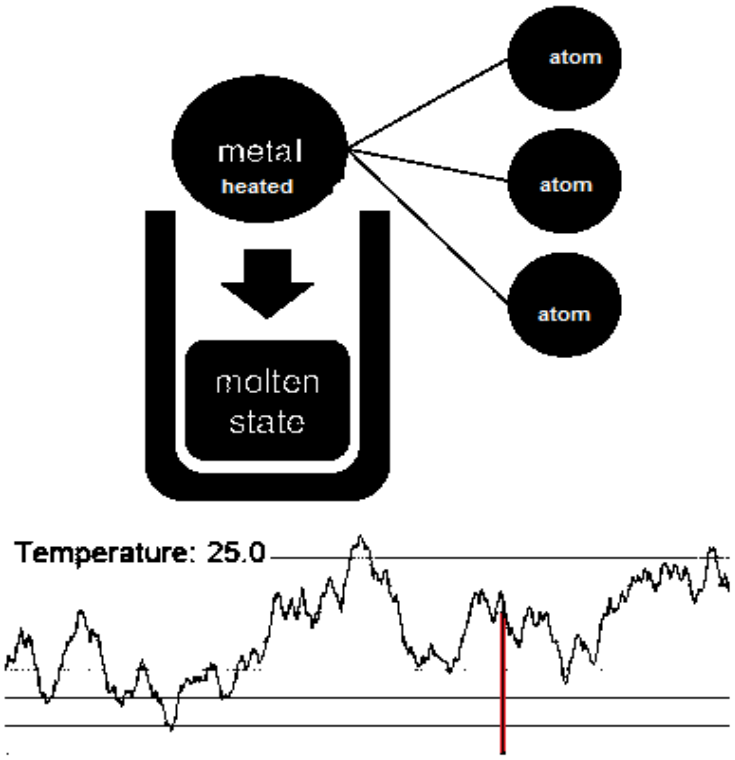


Figure 3.2. Simulated Annealing Representation

(Image courtesy: Engineering theory and practice, S.S.Rao, 2009)

The algorithm chooses a neighbor randomly and then determines whether to move on to the next state or not i.e, solution is generated by a suitable mechanism having the change in cost function. If cost function is reduced, the current solution is replaced by the generated neighbor. If increased, the generated neighbor replaces the current solution with an acceptance probability function is given by

$e^{(-[f[j]-f[i]]/T)}$ where $f[j]$ is generated state and $f[i]$ is the present state. The T is a control parameter, temperature. The small increase in f is more likely accepted than large increase, and also most of the generated neighbors are accepted when T is high, they are rejected if T approached zero. The initial temperature is always kept high so that it does not get trapped in local minima; the algorithm thus generates some neighbors at each temperature as it drops. Simulated annealing algorithm is guaranteed to get near optimal solution with a reasonable amount of computation time [116][117].

Pseudo-code for Simulated Annealing Algorithm

Generate a random solution Xm
For i=1 to δ (no of iterations)do
Generate a random solution X_1;
Generate a new solution X_1' for X_1;
If X_1' is better than Xm
Xmis updated with X_1'
Endif
Endfor
Return the best solution

3.3 The Ant colony algorithm

The Ant Colony Optimization (ACO) is a meta-heuristic optimization procedure that is, being inspired by the real ants foraging behaviour, and has become a popular technique for near exact optimization [118]. The main core behaviour here is the interaction between ants to find the food resources by means of a chemical substance called 'pheromone' track laid in the path to nest which facilitates them to find shortest paths between their nest and the food sources [119]. The selection of path depends on the amount of pheromone laid on the path makes the new ants to follow and add more pheromones. With a high probability of pheromone availability in shorter routes to food resources impresses that majority of ants to follow it. This indirect communication between ant is called "stigmergy", where the positive feedback concept is subjected to find best possible path, based on knowledge of resources by ants previously moved in that direction. The major advantage of Ant

Colony algorithm over other meta-heuristic algorithms is the problem instance may change dynamically.

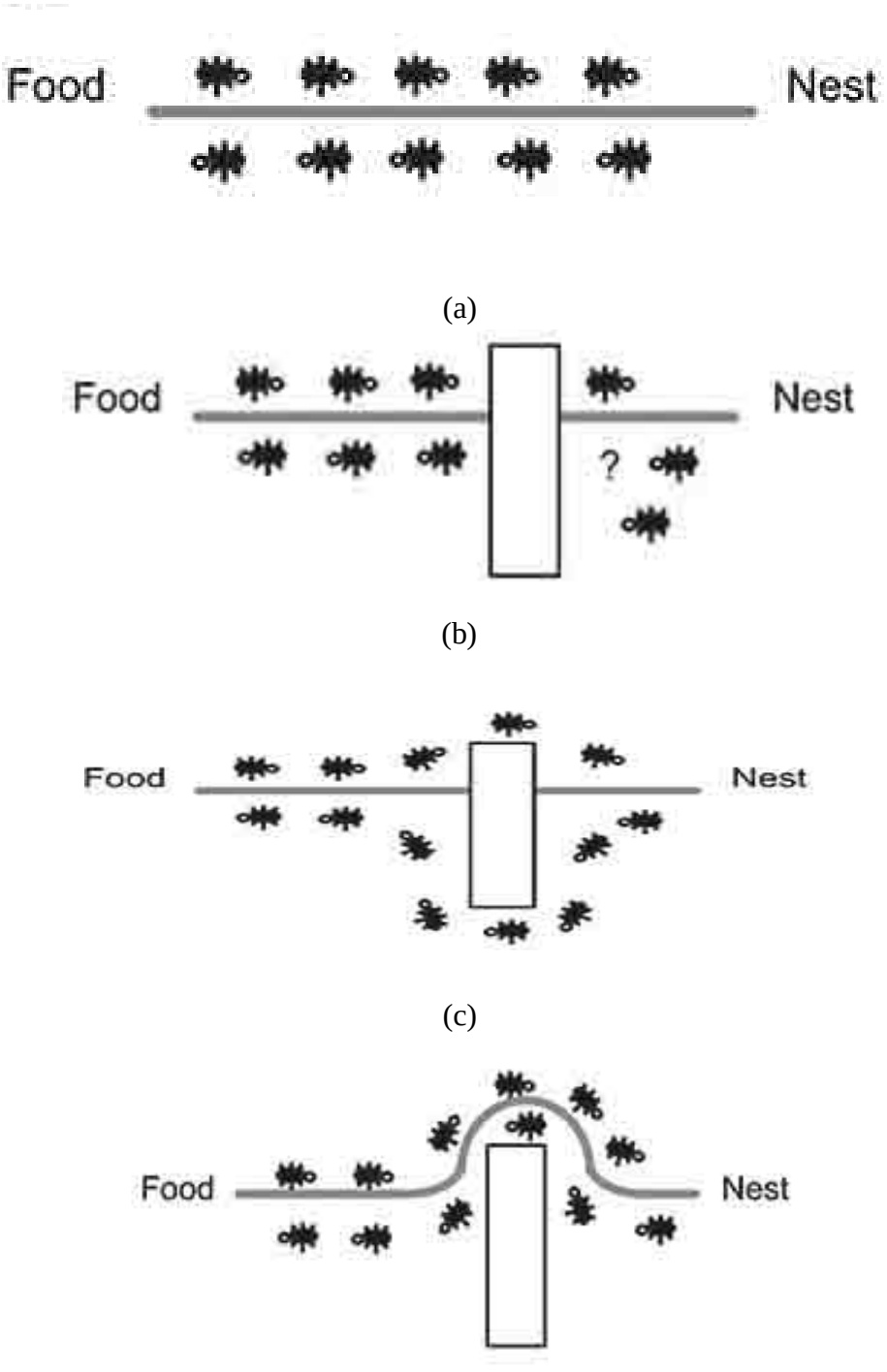


Fig. 3.3 Ants Behavioural Path

(*Image Courtesy: Article by Samer El-Khatib, Ukraine*)

Here, the decisions taken by roaming ants are vital and the knowledge of routing by ants are utilised in iterations to make the new optimal solution. This algorithm finds optimal paths based on the movements of ants searching for food. They search for food randomly in the distinct area, when they find the food source they go back to the colony leaving markers on the path i.e, pheromones; a chemical deposition. When other ants wondering in the area come across the pheromones, with a certain probability it is sure that they follow the path. If they find the pheromones evaporating, they drop their markers again when they go back. As other ants are directed to trail the chosen path, the chemical deposition in that direction becomes stronger and stronger. By the way, some ants randomly explore for food resources in the nearby and looking for shorter paths, if they find any obstacle they chose different paths as shown in the figure above. The near optimal solution can be obtained with the similar approach. As the food resources gets exhausted, the path is no longer colonized with high content of pheromones and slowly the chemical substance evaporates.

Since the ant colony optimization process is dynamic in its approach the algorithm is best suited for dynamic node allocation and can be designed for allocating tasks to ECUs in a similar fashion.

Pseudo-code for Ant Colony Algorithm

Select number of ECUs, tasks and iterations

Set min distance to traverse between ECUs

$$Distance=\sqrt{(x1-x2)^2+(y1-y2)^2}$$

Randomly select a path; find the distance

Generate different paths to reach destination

Among all the paths select the best path else start from primary place

Continue if more tasks to allocate

Else evaporate

The algorithm is used to follow some rules; the task allocation is considered for each ECU; busier the ECU, less chance of allocation; the more intense the ECU availability, greater the chosen probability; having allocated the task to ECU, the iteration completes. The best criterion to allocate tasks to available ECUs is to visit all the ECUs with minimal time.

3.4 The Genetic Algorithm

The Genetic Algorithm (GA) is a search based heuristic employing the evolution processes found in natural biology, introduced by John Holland in 1970 [120] and in Germany by Ingo Rechenberg [121]. Even though for many years the Genetic Algorithms have been studied, but implementation for a suitable problem is often seen as an art with efficient heuristic design. Often found in literature, the application of genetic algorithms is devoted to rather simple problems. Reasonable results have been regularly produced with the basic application of a genetic algorithm to simple problems, whereas for the larger problems, application of genetic algorithms often results in poor performance. This is largely due to natural characteristics of genetic search operation and also due to the relationships between a genetic representation and genetic operators a shown in figure 3.4.

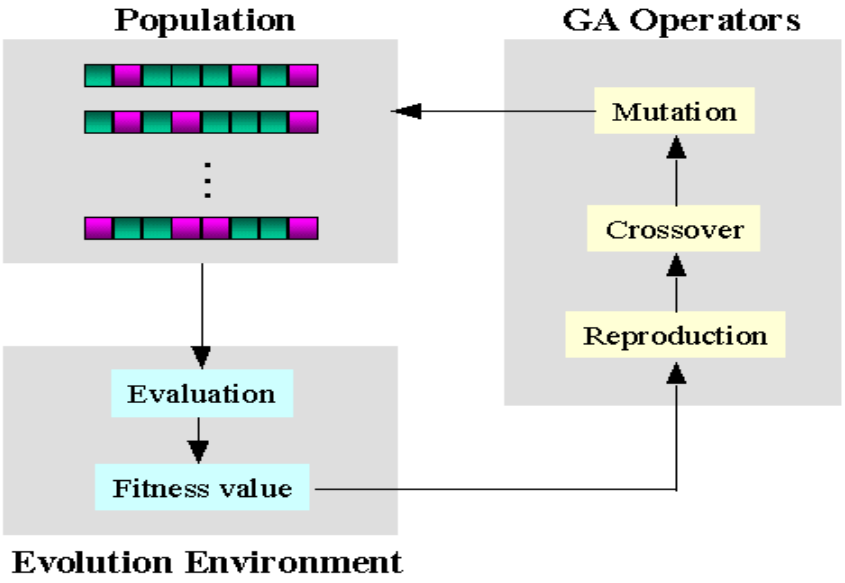


Figure 3.4 Genetic Algorithm Model

The genetic algorithm comes from the category of Evolutionary algorithms operating on given population of likely solutions to get them near some criteria or specification. Each evolutionary step in the process is called as either generation or reproduction, according to their level of fitness, from this new set of approximations are created, these are propagated through operators borrowed from natural genetics. The genetic algorithm is based on fundamental genetic operations on chromosomes such as selection method, crossover method and mutation method, to modify the

solution for obtaining appropriate resultant offspring. The best suited individuals which are selected for evolution process create the next generation.

In the search space, it selects many points at time and rapidly converges to possible solution or optimal solution. If the problem is too large, the genetic algorithm suffers from complex computations. It works on group of individuals for possible group of solutions instead of a single solution on the go. This gives scope for parallel processing for more complex and too large problems. It works on iterative procedure having all possible solutions for the current optimization problem. At every generation, the chosen individuals are decoded and are assessed based on fitness function. The performance of the individual in the population depends on the number of times it is chosen in a generation. The crossover is performed at single point or at multi-points to form new individuals from the selected individuals. The newly formed individuals are subjected to mutation that is some changes are caused by mistake during the process of gene copying from parents and this happens at random.

The pseudo code for the simple genetic algorithm is given below. It is required to create initial population over which the new generations are possible. Certain important considerations have to be looked for in task scheduling of multiprocessor or distributed embedded systems. The prominent are *dependency* and *resource sharing*. The dependent tasks have to be scheduled for execution on the same node or ECU. This will eliminate communication cost. The resource sharing by some tasks are preferably on priority basis and unfinished tasks may be shared among all nodes or ECUs. The scheduler should monitor and store in one queue accessed by all the processors or nodes.

Pseudo-code for Simple Genetic Algorithm

t:= 0; initialize time

Initialize ECUs N(t);

Evaluate availability of ECUs N(t);

While not complete do

t := t + 1;

N'(t) := select best of available ECUs N(t);

recombine N'(t);

mutate N'(t);

evaluate availability of N'(t);

N := allocate task N,N'(t);

End while

The three main elements that must be included in the chromosome format for scheduling algorithm are: the list of the tasks to be scheduled; the order of execution of tasks on a given node; the list of nodes over which these tasks are assigned.

The figure 3.5 below shows the chromosome format in which each column in the array represents gene. From the figure it shows that task 3 is executed by node 1; similarly node 2 executes task 4 and so on. The sequence of task execution is 3,4,1,2 and 5 over the nodes.

Tasks	3	4	1	2	5
Nodes	1	2	3	1	4

Figure 3.5: Chromosome format mapped with tasks on nodes.

In single point crossover, at randomly selected point, the nodes of parents swap the tasks as shown in figure 3.6.

Parent 1	Task	3	4	1	2	5
	Node	1	2	4	3	2
Parent 2	Task	4	6	5	3	7
	Node	2	4	3	5	6

(a) Before crossover

Child 1	Task	3	4	5	3	5
	Node	1	2	4	3	2
Child 2	Task	4	6	1	2	7
	Node	2	4	3	5	6

(b) After crossover

Figure 3.6: Single point crossover operation

An alternative is two point or multi-point crossover, at randomly selected point; the nodes of parents swap the tasks. The figure 3.7 shows two point crossover.

Parent 1	Task	3	4	1	2	5
	Node	1	2	4	3	2
Parent 2	Task	4	6	5	3	7
	Node	2	4	3	5	6

(a) Before crossover

Child 1	Task	4	4	1	3	5
	Node	1	2	4	3	2
Child 2	Task	3	6	5	2	7
	Node	2	4	3	5	6

(b) After crossover

Figure 3.7: Two point crossover operation

During evolution there is a possibility that the genetic material can change randomly. In genetic algorithm, this random deformation of genes with certain probability is mutation. This property has a positive effect on offspring that it is not going to stuck in local optima. The mutation rate is the probability of change in cell. The selection of cell for mutation indicates either task or the node selection is randomly changed. The figure 3.8 below shows the simple mutation process.

Tasks	3	4	1	2	5
Nodes	1	2	3	1	4

(a) Chromosome before mutation

Tasks	3	4	5	2	5
Nodes	1	2	3	1	4

(b) Chromosome after mutation

Figure 3.8: Simple mutation process

3.5 Proposed Method

The above algorithms present task allocation on an ECU considering uniprocessor architecture whereas the Time Triggered Architecture is a multiprocessor architecture having constraints of fault tolerant design. Thus certain important considerations have to be looked for in task scheduling on multiprocessor or distributed embedded systems. The prominent parameters are dependency and resource sharing. In GA, instead of single crossover and single mutation, a multiple crossover and multiple mutation is performed to meet the above requirements. This is said to be Dynamic Genetic Algorithm (DGA).

In addition to multiple crossovers and multiple mutations, a multiple warehouse (***Task generator***) condition is added here to solve the problem more efficiently. Each salesman visits a set of cities originating from a warehouse and returning back and also exactly one salesman visits each city. Thus, a simple genetic algorithm is combined with Multi Warehouse Multiple Travelling Salesmen Problem (MWMTSP) to schedule on multiprocessor system like TTA. MWMTSP with variable number of salesmen using GA finds a near optimal solution referred in our work hereafter as Adaptive Dynamic Genetic Algorithm (ADGA).

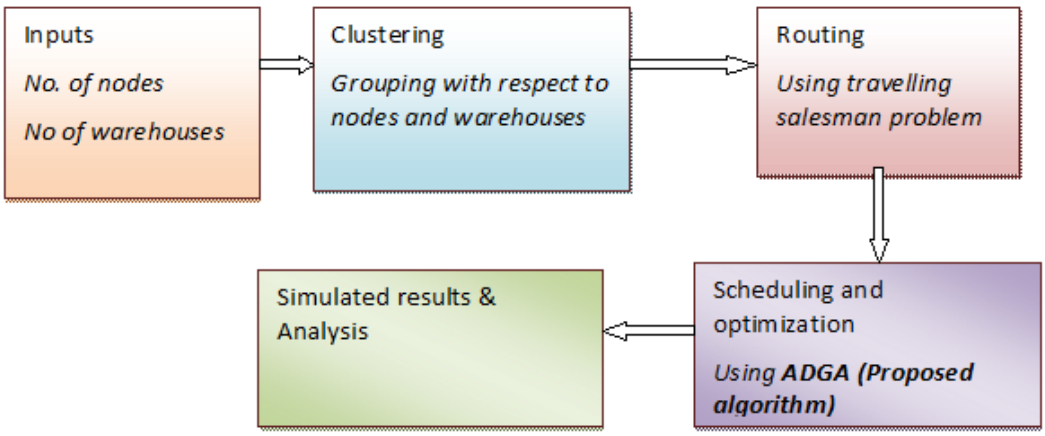


Figure 3.9 Block Diagram of Proposed Method

MWMTSP is a more challenging problem than simple MTSP also it is NP-hard, means it is difficult to solve with exact procedures. This is dealt more efficiently and effectively by applying Genetic algorithm to the problem. The figure 3.9 above

illustrates the block diagram of the proposed method. Grouping is done with respect to nodes and warehouses in random fashion with Euclidean distance calculation. The routing problem is considered from travelling salesman problem. The intent of routing here is to implement parallelism and reduce the timing cost. The scheduling and optimization is done using proposed Adaptive Dynamic Genetic algorithm. The results are evaluated with respect to number of nodes, number of clusters or routes and best solution. A relative scrutiny of the proposed heuristic technique is performed with respect to other techniques.

Pseudo-code for Proposed Algorithm

Initialization

t= 0;

N (t) = ECUs;

$W_{i...n}$ = *Warehouses;*

Evaluate availability of $W_{i...n}$ *and N (t);*

For

$W_i \neq W_n$

Create a population

While not complete do

Run GA % GA = Genetic Algorithm

t := t + 1;

N'(t) := select best of available ECUs N(t);

crossover N'(t);

mutate N'(t);

select N'(t);

N_s *:= allocate task N,N'(t);*

Remove Scheduled Task

End while

End For

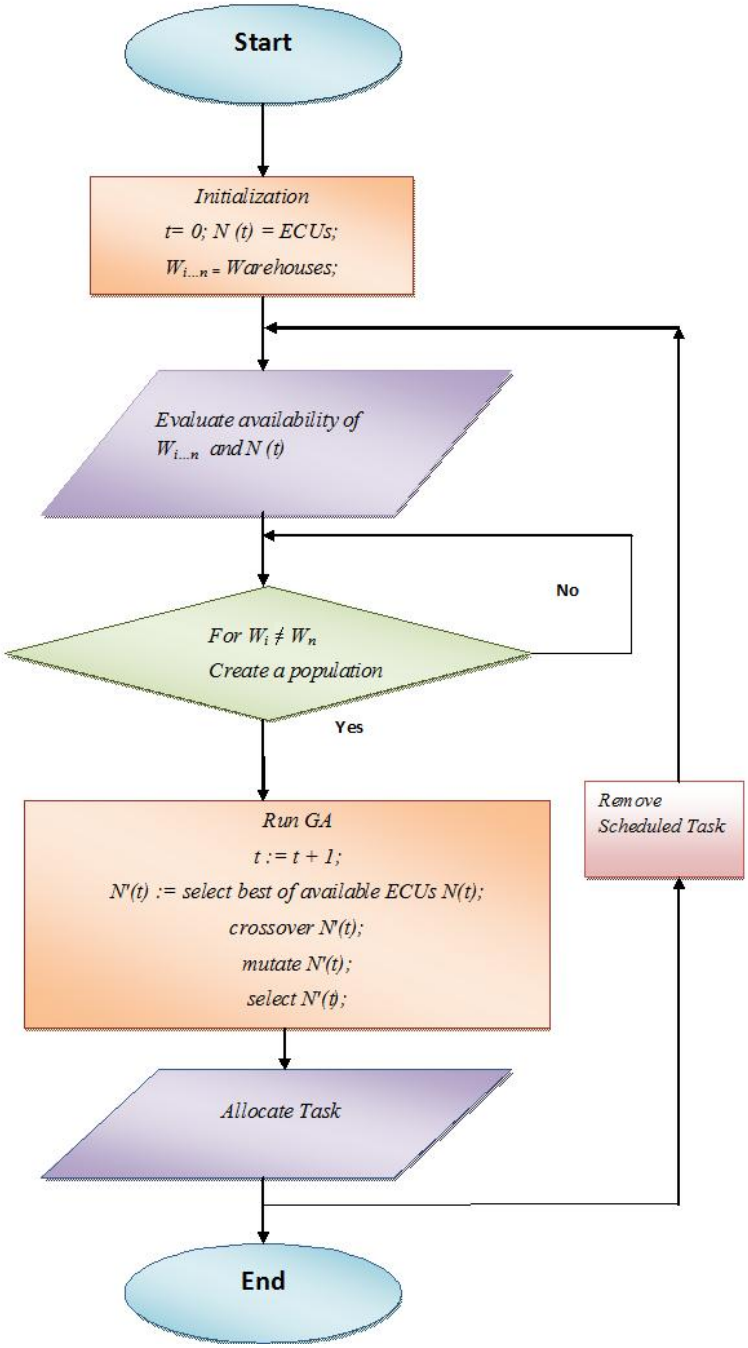


Figure 3.10. Process Flow Diagram

In this work we follow the functionality that allows finding best solution of task allocation on ECUs (minimum sum of all tour lengths). MWMTSP solution is such that there are no two routes based at the same warehouse.

3.6 Methodology

The Methodology is given below as a step by step procedure to solve the considered Model.

a. Initialization of the Significant Attributes:

As per the proposed model, Significant attributes of the process i.e. number of inputs and warehouses required are applied as inputs.

b. Implementation of algorithms:

The optimization algorithms discussed in proposed model and also existing algorithms are implemented based on their logical computations using **MATLAB** technical computing language ***(R2010a version and above)*** using toolboxes Genetic algorithm, Optimization, fixed point and Statistics and Machine Learning on Intel Core i5 PC at 2.5GHZ with a total physical memory of 4 GB RAM.

c. Testing:

The proposed algorithm tested on the given inputs and important features from extensive testing have been extracted.

d. Analysis:

Using the results of the tests, an assessment of the algorithm is made. Therefore an analysis is carried out on the results obtained from existing and proposed algorithms. The results are compared based on best solution and also a graphical representation is presented. Any relative strengths or weaknesses of algorithms should be found.

e. Modified algorithm:

Based on the analysis, one or more improvements to existing recent techniques have become apparent. Thus a new algorithm is devised namely Adaptive Dynamic Genetic Algorithm (ADGA) which demonstrates the improvements.

f. Test modified algorithm:

The improved algorithm is evaluated on the basis of the requirements to analyze the best optimum solution in an easiest way.

g. Outcome:

Finally the research project resulted in the creation of a new algorithm that combined ideas from the existing optimization techniques and was useful to attain the motto.