CHAPTER 4

DIFFERENTIAL EVOLUTION WITH VARIABLE NEIGHBORHOOD SEARCH FOR SCHEDULING IN MULTIPROCESSOR SYSTEMS

4.1 NEED FOR SCHEDULING IN MULTIPROCESSOR SYSTEMS

Scheduling for a set of dependent and independent task that execute in parallel on a set of processors is a complex problem. Multiprocessors have widely been used in parallel computation. Appropriate scheduling strategies are required to reach a higher efficiency which in turn reduces the makespan time. The make span of a schedule can be defined as the time it takes from the instant the first task begins execution to the instant at which the last task completes execution.

Another objective of the scheduling is to provide precedence relationship among the tasks when allocating them to processors. In this context there are a lot of issues that should be addressed in terms of dependent or independent tasks, task graph produced randomly, homogeneity and heterogeneity of the multiprocessor systems.
Scheduling approaches can be classified according to the arrival time of tasks into static and dynamic. In static scheduling, all tasks to be scheduled are ready to be dealt with. In other words, no tasks arrive later. On the other hand, dynamic scheduling contains tasks with unfixed arrival times. In other words, tasks can arrive at some known or unknown future time. Scheduling approaches can also be classified according to the availability of related information into deterministic or stochastic scheduling. In deterministic category, all related information is either known or with certainty, such that tasks, processing times, and any other information, are available at the beginning, or arrive at a known future time.

4.2 SCHEDULING IN MULTIPROCESSOR SYSTEMS

Edwin S.H (1994) formalized the multiprocessor scheduling problem, a (homogeneous) multiprocessor system and a parallel program are defined. A (homogeneous) multiprocessor system is composed of a set of m identical processors. Each processor can execute at the most one task at a time and task preemption is not allowed. A set of computational tasks can be represented by a directed acyclic task graph. The task graph G = (V, E) has a set of nodes V and a set of directed edges E which connects the nodes to each other.

V is a set of tasks to be executed, V = \{T_1, T_2... T_m \}.

The edges are represented by E = \{e_{ij}\} where e_{ij} is the edge between two tasks T_i and T_j and it specifies that T_i should be completed before T_j can be started. The notation T_i \geq T_j is used for this purpose and it shows that T_i is the predecessor of and T_j is the successor of T_i.
A set of predecessors of tasks $T_i$ is denoted by $\text{PRED} (T_i)$. A set of successors of tasks $T_i$ is denoted by $\text{SUCC} (T_i)$. An arc from $T_1$ to $T_2$ represents the fact that at the end of its execution, $T_1$ sends a message whose contents are required by $T_2$ to start execution. In this case, $T_1$ is said to be an immediate predecessor of $T_2$, and $T_2$ itself is said to be an immediate successor of $T_1$. The height of a task is represented by the equation (4.1)

$$\text{Height}(T_i) = \begin{cases} 
0 & \text{if } \text{PRED}(T_i) = 0 \\
1 + \max \text{ Height}(T_j) & \text{Tj c PRED}(T_i) \text{ otherwise}
\end{cases} \quad (4.1)$$

![Diagram of a Directed Acyclic Graph (DAG)](image)

**Figure 4.1 An example for DAG**

By definition, for any two tasks $T_i$ and $T_j$, if $T_i$ is an ancestor of $T_j$, then $T_i$ has greater value of height than $T_j$. This height function conveys the precedence relations between the tasks. In fact, if task $T_i$ is an ancestor of task $T_j$ (i.e., if $T_i$ must be executed before $T_j$), then height $(T_i) <$ height $(T_j)$. If there is no path between the two tasks, then there is no precedence relation between the two tasks. The simple DAG with 8 tasks is given in Figure 4.1.
Figure 4.2 A legal schedule for the DAG

In multiprocessor system the problem of optimal scheduling a task graph with P processors is to assign the computational tasks to the processors so that the precedence relations are maintained and makespan of the schedule is minimized. The finishing time (FT) of the schedule is defined as the time that the last task is completed. A Directed Acyclic Graph with eight tasks according to their height and execution time is shown in Figure 4.2. Execution time is the time needed for a task to execute.

4.3 DIFFERENTIAL EVOLUTION WITH VARIABLE NEIGHBORHOOD SEARCH AND RANDOM SCALE FACTOR

4.3.1 Introduction

Optimization is everywhere, from engineering design to financial markets, in our daily activity to planning our holidays, and computer sciences to industrial applications. It was always intend to maximize or minimize something. An organization wants to maximize its profits, minimize costs, and maximize performance. Even when to plan the holidays, the aim is to maximize the enjoyment with least cost (or ideally free). In fact, the aim is to constantly searching for the optimal solutions to every problem, though it is not necessarily able to find such solutions. Optimization is mainly classified as heuristic and
metaheuristic. Heuristics is a solution strategy by trial-and-error to produce acceptable solutions to a complex problem in a reasonably practical time. The complexity of the problem of interest makes it impossible to search every possible solution or combination, the aim is to find good, feasible solutions in an acceptable timescale. There is no guarantee that the best solutions can be found, and even do not know whether an algorithm will work and why if it does work. The idea is that an efficient but practical algorithm that will work most of the time and be able to produce good quality solutions. Among the found quality solutions, it is expected that some of them are nearly optimal, though there is no guarantee for such optimality. From the literature it was noticed that more and more metaheuristic algorithms are being developed. Such a diverse range of algorithms necessitates a system summary of various metaheuristic algorithms.

In addition, meta-heuristic methods are easy to implement and can be combined with other algorithms. The characteristics of the objective function and/or constraints are inconsequential to the success of those methods. However, meta-heuristic algorithms have parameters to be tuned, an adjustment that is commonly accomplished by trial-and-error experiments as well as the skill of the user. Consequently, an efficient algorithm with fewer parameters to be adjusted is always more favorable. Enhancing performance by accelerating and/or improving the solution quality is an ongoing task for most meta-heuristic algorithms. Various modified versions seeking enhanced results have been suggested in the literature. However, many of those adaptive methods may not succeed depending upon the particular problems and/or additional parameters needing to be tuned.

In 1995, Price and Storn proposed a new floating point encoded evolutionary algorithm for global optimization and named it DE owing to a special kind of differential operator, which they invoked to create new offspring
from parent chromosomes instead of classical crossover or mutation. It has been proven a robust global optimizer and has been successfully applied to many global optimization problems with superior performance in both widely used benchmark function and real-world applications.

The DE algorithm is an evolutionary algorithm with less stochastic approach to problem solving than classical evolutionary algorithms. DE has the potential of simple in structure, fast convergence property and quality of solution. A hybrid solution methodology combination of DE and variable neighborhood search (VNS) was applied to improve the quality of solution and convergence speed. Differential evolution (DE) is first introduced to find the locality of the solution, and then VNS is applied to tune the solution. Later DE with Random Scale Factor (DE-RSF) is applied for solving the above problem.

The aim of this chapter is to provide a concise survey and application of one of the newest metaheuristic methods tuned with deterministic methods to find solution to assign the computational tasks to the processors so that the precedence relations are maintained and makespan of the schedule is minimized.

4.4 DIFFERENTIAL EVOLUTION: AN OVERVIEW

The Differential Evolution (DE), proposed by Storn and Price (1995, 1997) may be also seen as a simple real-coded GA. The first written article on DE appeared as a technical report in 1995. Since then, DE has proven itself in competitions like the IEEE’s International Contest on Evolutionary Optimization (ICEO) in 1996 and 1997 respectively.
In DE community, the individual trial solutions (which constitute a population) are referred as parameter vectors or genomes. DE operates through the same computational steps as employed by a standard EA. However, unlike traditional EAs, DE employs difference of the parameter vectors to explore the objective function landscape. In this respect, it owes a lot to its two ancestors namely – the Nelder-Mead algorithm (1965, 2003), and the Controlled Random Search (CRS) algorithm (1977), which also relied heavily on the difference vectors to perturb the current trial solutions. Like other population-based search techniques, DE generates new points (trial solutions) that are perturbations of existing points, but these deviations are neither reflections like those in the CRS and Nelder-Mead methods, nor samples from a predefined probability density function, like those in Evolutionary Strategies (ES)(1966,2003). Instead, DE perturbs current generation vectors with the scaled difference of two randomly selected population vectors. To produce a trial vector in its simplest form DE adds the scaled, random vector difference to a third randomly selected population vector. In the selection stage, the trial vector competes against the population vector of the same index. Once the last trial vector has been tested the survivors of all the pair wise competitions become permanent for the next generation in the evolutionary cycle. In the following Sections, we discuss each of these steps in sufficient details. DE is a simple evolutionary algorithm. It works through a simple cycle of stages, presented in Figure 4.3.

![Figure 4.3: The main stages of differential evolution algorithm](image)
4.4.1 Classical DE – How does it Work?

Like any other evolutionary algorithm, DE also starts with a population of NP, D-dimensional search variable vectors. Subsequent generations in DE is represented by discrete time steps like \( t = 0, 1, 2, \ldots, t, \ t+1, \ldots \). Since the vectors are likely to be changed over different generations. So it was adopted the following notation for representing the \( i \)th vector of the population at the current generation (i.e., at time \( t = t \)) as

\[
\vec{X}_i(t) = [x_{i,1}(t), x_{i,2}(t), x_{i,3}(t), \ldots, x_{i,D}(t)]
\]  (4.2)

These vectors are referred in literature as “genomes” or “chromosomes”. DE is a very simple evolutionary algorithm. For each search-variable, there may be a certain range within which value of the parameter should lie for better search results. At the very beginning of a DE run or at \( t = 0 \), problem parameters or independent variables are initialized somewhere in their feasible numerical range. Therefore, if the \( j \)th parameter of the given problem has its lower and upper bound as \( x_j^L \) and \( x_j^U \), respectively, then the \( i \)th population members was initialized as

\[
x_{i,j}(0) = x_j^L + \text{rand}(0, 1) \cdot (x_j^U - x_j^L),
\]  (4.3)

Where \( \text{rand}(0, 1) \) is a uniformly distributed random number lying between 0 and 1.

Now in each generation (or one iteration of the algorithm) to change each population member \( \vec{X}_i(t) \) (say), a Donor vector \( \vec{V}_i(t) \) is created. It is the method of creating this donor vector, which demarcates between the various DE schemes. However, in this chapter we discuss one such specific mutation strategy known as DE/rand/1. In this scheme, to create \( \vec{V}_i(t) \) for each \( i \)th
member, three other parameter vectors (say \(r_1, r_2\), and \(r_3\) th vectors) are chosen in a random fashion from the current population. Next, a scalar number \(F\) scales the difference of any two of the three vectors and the scaled difference is added to the third one when we obtain the donor vector \(\tilde{V}_i(t)\). We can express the process for the \(j\)th component of each vector as

\[
v_{i,j}(t+1) = x_{r_{1,j}}(t) + F \cdot (x_{r_{2,j}}(t) - x_{r_{3,j}}(t))
\]  

(4.4)

Next, to increase the potential diversity of the population a crossover scheme comes to play. DE can use two kinds of cross over schemes namely “Exponential” and “Binomial”. The donor vector exchanges its “body parts”, i.e., components with the target vector \(\tilde{X}_i(t)\) under this scheme. In “Exponential” crossover, it was chosen an integer \(n\) randomly among the numbers \([0, D-1]\). This integer acts as starting point in the target vector, from where the crossover or exchange of components with the donor vector starts. It was also chosen that another integer \(L\) from the interval \([1, D]\). \(L\) denotes the number of components: the donor vector actually contributes to the target. After a choice of \(n\) and \(L\) the trial vector:

\[
\tilde{U}_i(t) = [u_{i,1}(t), u_{i,2}(t), \ldots, u_{i,D}(t)]
\]

(4.5)
is formed with

\[
u_{i,j}(t) = v_{i,j}(t) \text{ for } j = \langle n \rangle_D, \langle n + 1 \rangle_D, \ldots, \langle n - L + 1 \rangle_D = x_{i,j}(t)
\]

(4.6)

where the angular brackets \(\langle \rangle_D\) denote a modulo function with modulus \(D\). The integer \(L\) is drawn from \([1, D]\) according to the following pseudo code.

\[
L = 0;
\]
do

{
\[ L = L + 1; \]

\[ \text{while (rand (0,1) < CR ) AND ( L < D ));} \]

Hence in effect probability \((L > m) = (CR)^{m-1}\) for any \(m > 0\). \(CR\) is called “Crossover” constant and it appears as a control parameter of DE just like \(F\). For each donor vector \(V\), a new set of \(n\) and \(L\) must be chosen randomly as shown above. However, in “Binomial” crossover scheme, the crossover is performed on each of the \(D\) variables whenever a randomly picked number between 0 and 1 is within the \(CR\) value. The scheme may be outlined as

\[ u_{i,j}(t) = u_{i,j}(t) \text{if rand ( 0,1 ) < CR, } x_{i,j}(t) \]  \hspace{1cm} (4.7)

In this way for each trial vector \(\tilde{X}_i(t)\) an offspring vector \(\tilde{U}_i(t)\) is created. To keep the population size constant over subsequent generations, the next step of the algorithm calls for “selection” to determine which one of the target vector and the trial vector will survive in the next generation, i.e., at time \(t = t + 1\). DE actually involves the Darwinian principle of “Survival of the fittest” in its selection process which may be outlined as

\[ \begin{align*}
\tilde{X}_i(t+1) &= \tilde{U}_i(t) \text{ if } f(\tilde{U}_i(t)) \leq f(\tilde{X}_i(t)), \\
\tilde{X}_i(t) & \text{ if } f(\tilde{X}_i(t)) < f(\tilde{U}_i(t))
\end{align*} \]  \hspace{1cm} (4.8)

where \(f(x)\) is the function to be minimized. So if the new trial vector yields a better value of the fitness function, it replaces its target in the next generation; otherwise the target vector is retained in the population. Hence the population either gets better (w.r.t the fitness function) or remains constant but never deteriorates. The \(DE/rand/1\) algorithm is outlined below:
4.4.1.1 DE- Procedure

**Input:** Randomly initialized position and velocity of the particles: $\pi_i(0)$

**Output:** Position of the approximate global optima $\tilde{x}^*$

**Begin**

Initialize population;
Evaluate fitness;

For $i = 0$ to max-iteration do

**Begin**
Create Difference-offspring;
Evaluate fitness;
If an offspring is better than its parent
Then replaces the parent by offspring in the next generation;
End if;
End For;

End.

Differential evolution (DE) was designed as a stochastic direct search method to handle non-differentiable, nonlinear and multimodal cost functions. DE starts with an initial population $NP$ that is generated by adding normally distributed random deviations to the nominal solution. The initial population utilizes $NP$ $D$-dimensional parameter vectors for each generation, whereas $D$ represents dimension of the problem. The $i$th individual in the population is represented by following expression:

$$\tilde{X}_i = \left[ x_{i1}, x_{i2}, x_{i3}, ..., x_{iD} \right]$$ (4.9)

The initial population should better cover the entire search space as much as possible by uniformly randomizing individuals within the search space.
constrained by the prescribed minimum and maximum parameter bounds. The 
i-th individual of the population for current generation $G$ is given by following 
expression

$$x_{ij,G} = x_{ij}^{\text{min}} + \text{rand}(0,1) \cdot (x_{ij}^{\text{max}} - x_{ij}^{\text{min}})$$

(4.10)

where $j = 1, 2 \ldots, D$ and rand(0,1) represents a uniformly distributed random 
variable within the range $[0, 1]$. $x_{ij}^{\text{max}}$ and $x_{ij}^{\text{min}}$ represents higher and lower 
bounds of the search space. The initial population is subjected to following 
different operations.

4.4.1.2 Mutation

For each target vector $X_{i,G}$, a mutant vector is generated as follows:

$$V_{i,G,1} = x_{i1,G} + F(x_{i2,G} - x_{i3,G})$$

(4.11)

Where random indexes, $r1, r2, r3 \in \{1, 2 \ldots NP\}$ are integers that is 
mutually different to each other and to the running index $i$. $F$ is a real and 
constant factor which controls the amplification of different variation 
$(x_{i2,G} - x_{i3,G})$. It is the method of creating this donor vector that differentiates 
one DE scheme from another. Equation (4.10) is known as “DE/rand/1” strategy 
and throughout this paper the strategy “DE/rand/1” is followed.

4.4.1.3 Crossover

To increase the diversity of the perturbed vectors, crossover is 
introduced. It results in formation of a trial vector which is represented as:
\[ u_{i,G+1} = \left\{ u_{i,G+1}^{u_1}, u_{i,G+1}^{u_2}, u_{i,G+1}^{u_3}, \ldots, u_{i,G+1}^{u_D} \right\} \]  \hspace{1cm} (4.12)

The crossover is applied to each pair of target vector, \( X_{i,G} \) and mutant vector, \( V_{i,G} \) to form a trial vector. It is performed by following expression:

\[
u_{j,G+1} = \begin{cases} v_{j,G} & \text{if} \ (\text{randb}(j) \leq CR) \\ x_{j,G} & \text{if} \ (\text{randb}(j) > CR), \ \text{where} \ j = 1, 2, \ldots, D \end{cases}
\]  \hspace{1cm} (4.13)

Where \( j = 1, 2, \ldots, D \) and \( \text{randb}(j) \) is the \( j \)th evaluation of a uniform random number generator with outcome \( \in [0, 1] \). \( CR \) is the crossover constant \( \in [0, 1] \).

### 4.4.1.4 Selection

To decide whether or not \( U_{i,G+1} \) should become a member of generation \( G+1 \), it is compared to the target vector \( X_{i,G} \) using the greedy criterion. It is given as follows:

\[
x_{i,G+1} = \begin{cases} u_{i,G+1} & \text{if} \ f(u_{i,G+1}) < f(x_{i,G}) \\ x_{i,G} & \text{otherwise} \end{cases}
\]  \hspace{1cm} (4.14)

Set the generation number for \( G = G + 1 \). Repeat the above steps until a stopping criterion is met, usually a maximum number of iterations (generations), \( G_{\text{max}} \).

### 4.4.2 The Complete DE Family of Storn and Price

Actually, it is the process of mutation, which demarcates one DE scheme from another. In the former section, we have illustrated the basic steps
of a simple DE. The mutation scheme here uses a randomly selected vector $\bar{X}_{r_1}$ and only one weighted difference vector $F(\bar{X}_{r_2} - \bar{X}_{r_3})$ is used to perturb it. Hence, in literature the particular mutation scheme is referred to as DE/rand/1. It can now have an opportunity how to name different DE schemes. The general convention used, is DE/x/y.DE stands for DE, $x$ represents a string denoting the type of the vector to be perturbed (whether it is randomly selected or it is the best vector in the population with respect to fitness value) and $y$ is the number of difference vectors considered for perturbation of $x$. Below we outline the other four different mutation schemes, suggested by Price et al. (1997)

4.4.3 Scheme DE/rand to best/1

DE/rand to best/1 follows the same procedure as that of the simple DE scheme illustrated earlier. The only difference being that, now the donor vector, used to perturb each population member, is created using any two randomly selected member of the population as well as the best vector of the current generation (i.e., the vector yielding best suited objective function value at($t = t$)). This can be expressed for the $i$th donor vector at time $t = t+1$ as

$$V_i(t+1) = \bar{X}_i(t) + \lambda (\bar{X}_{best}(t) - \bar{X}_i(t)) + F(\bar{X}_{r_2}(t) - \bar{X}_{r_3}(t))$$  \hspace{1cm} (4.15)

where $\lambda$ is another control parameter of DE in $[0,2]$, $\bar{X}_i(t)$ is the target vector and $\bar{X}_{best}(t)$ is the best member of the population regarding fitness at current time step $t = t$. To the number of control parameters a usual choice is to put $\lambda = F$.

4.4.4 Scheme DE/best/1

In this scheme everything is identical to DE/rand/1 except the fact that the trial vector is formed as
\[
\vec{V}_i(t+1) = \vec{X}_{best}(t) - F \cdot \left( \vec{X}_{r_1}(t) - \vec{X}_{r_2}(t) \right) 
\] (4.16)

Here the vector to be perturbed is the best vector of the current population and the perturbation is caused by using a single difference vector.

### 4.4.5 Scheme DE/best/2

Under this method, the donor vector is formed by using two difference vectors as shown below:

\[
\vec{V}_i(t+1) = \vec{X}_{best}(t) + F \cdot \left( \vec{X}_{r_1}(t) + \vec{X}_{r_2}(t) - \vec{X}_{r_3}(t) - \vec{X}_{r_4}(t) \right) 
\] (4.17)

Owing to the central limit theorem the random variations in the parameter vector seems to shift slightly in to the Gaussian direction which seems to be beneficial for many functions.

### 4.4.6 Scheme DE/rand/2

Here the vector to be perturbed is selected randomly and two weighted difference vectors are added to the same to produce the donor vector. Thus for each target vector, a totality of five other distinct vectors are selected from the rest of the population.

The process can be expressed in the form of an equation as

\[
\vec{V}_i(t+1) = \vec{X}_{r_1}(t) + F_1 \cdot \left( \vec{X}_{r_2}(t) - \vec{X}_{r_3}(t) \right) + F_2 \cdot \left( \vec{X}_{r_4}(t) - \vec{X}_{r_5}(t) \right) 
\] (4.18)

Here \( F_1 \) and \( F_2 \) are two weighing factors selected in the range from 0 to 1. To reduce the number of parameters choose \( F_1 = F_2 = F \). The flowchart for basic DE algorithm is given in Fig 4.4
Figure 4.4: Flowchart for basic DE Algorithm
4.4.7 Summary of all Schemes

In 2001 Storn and Price suggested total ten different working strategies of DE and some guidelines in applying these strategies to any given problem. These strategies were derived from the five different DE mutation schemes outlined above. Each mutation strategy was combined with either the “exponential” type crossover or the “binomial” type crossover. This yielded $5 \times 2 = 10$ DE strategies, which are listed below.

1. DE/best/1/exp
2. DE/rand/1/exp
3. DE/rand-to-best/1/exp
4. DE/best/2/exp
5. DE/rand/2/exp
6. DE/best/1/bin
7. DE/rand/1/bin
8. DE/rand-to-best/1/bin
9. DE/best/2/bin
10. DE/rand/2/bin

The general convention used above is again DE/x/y/z, where DE stands for DE x represents a string denoting the vector to be perturbation, y is the number of difference vectors considered for perturbation of x, and z stands for the type of crossover being used (exp :exponential; bin-binomial)

4.5 VARIABLE NEIGHBORHOOD SEARCH: AN OVERVIEW

The Variable Neighborhood Search algorithm (VNS) is a relatively recent metaheuristic which relies on iteratively exploring neighborhoods of
growing size to identify better local optima. More precisely, VNS escapes from
the current local minimum \( x^* \) by initiating other local searches from starting
points sampled from a neighborhood of \( x^* \) which increases its size iteratively
until a local minimum better than the current one is found. These steps are
repeated until a given termination condition is met.

4.5.1 Rules

It is denoted with \( \mathcal{N}( k = l, \ldots, k_{\text{max}} ) \), a finite set of preselected
neighborhood structures, and with \( \mathcal{N}(x) \) the set of solutions in the \( k^{th} \)
neighborhood of \( x \). (Most local search heuristics use one neighborhood structure,
i.e., \( k_{\text{max}} = 1 \)). The stopping condition may be e.g., maximum CPU time allowed,
maximum number of iterations, or maximum number of iterations between two
improvements. Often successive neighborhood \( \mathcal{N} \) will be nested. Observe that
point \( x' \) is generated in random steps in order to avoid cycling, which might
occur if any deterministic rule was used.

4.5.2 Steps of Basic VNS

Initialization: Select the set of neighborhood structures \( \mathcal{N} \)
\( k = l, \ldots, k_{\text{max}} \), that will be used in the search; find an initial solution \( x \);
choose a stopping condition.

Repeat the following until the stopping condition is met:

1. Set \( k \leftarrow 1 \); (2) Until \( k = k_{\text{max}} \), repeat the following steps:

2. Shaking. Generate a point \( x' \) at random from the \( k^{th} \) neighborhood
   of \( x(x' \in \mathcal{N}(x)) \)
3. **Local search.** Apply some local search method with \( x' \) as initial solution; denote with \( x'' \) the so obtained local optimum;

4. **Move or not.** If this local optimum is better than incumbent, move there \( (x \leftarrow x'') \), and continue the search with \( N(k \leftarrow 1) \)

5. otherwise, set \( k \leftarrow k + 1 \)

As a local optimum within some neighborhood is not necessarily one within another, change of neighborhoods can be performed during the local search phase also. In some cases, as when applying VNS to graph theory, the use of many neighborhoods in the local search is crucial.

VNS has been applied to a wide variety of problems both from combinational and continuous problems were based on a particular problem structure. In continuous location-allocation problem the neighborhoods are defined according to the meaning of problem. In bilinearly constrained problems the neighborhoods are defined in terms of the applicability of the successive linear programming approach, where the problem can be partitioned so that fixing the variables in either in set yields a linear problems, more precisely, the neighborhoods of size \( k \) are defined as the vertices of the LP polyhedral that are \( k \) pivots away from the current vertex. However, none of the early applications of the early applications of VNS to continuous problems in general form.

The first VNS algorithm targeted at problems with fewer structural requirements, namely, box constrained NLP. Since the problem is assumed to be box constrained, the neighborhoods arise naturally as hyper rectangles of growing size centered at the current local minimum \( x'' \).
4.5.3 The VNS Algorithm.

Input maximum number of neighborhoods $k_{\text{max}}$, number of local searches in each neighborhood $L$.

Loop

Set $k \leftarrow 1$, pick random point $\tilde{x}$, and perform a local search to find a local minimum $x^*$.

While $k \leq k_{\text{max}}$ do

Consider a neighborhood $\mathcal{N}(x^*)$ of $x^*$ such that $\forall k > 1, \mathcal{N}(x^*) \supseteq \mathcal{N}_k(x^*)$.

for $i=1$ to $L$ do

Sample a random point $\tilde{x}$ from $\mathcal{N}(x^*)$.

Perform a local search from $\tilde{x}$, to find a local minimum $x'$.

If $x'$ is better than $x^*$, set $x^* \leftarrow x'$, $k \leftarrow 0$ exit the FOR loop.

End for

Set $k \leftarrow k + 1$.

Verify termination condition; if true, exit.

end while

end loop

4.6 IMPLEMENTATION of DE-VNS

Step 1) Parameter setup

Initialize the number of generating units $N$ and Population size $NP$. Specify minimum and maximum capacity of each generator $P_{\text{min}}$ and $P_{\text{max}}$ respectively. Initialize DE parameters like Crossover Probability $CR$, Scaling Factor $F$. Set generation count, $G = 0$. 
Step 2) Initialization of the population

For a population size NP and dimension D, an initial vector $X_{ij,G}$ is randomly generated. D represents the number of decision variables to be optimized. In ELD problem D is the number of generating units considered. $X_{ij}$ is the real power value of $j$th unit of the $i$th population randomly generated within the operating limits.

Step 3) Evaluation of fitness function

Evaluate the fitness value of each individual vector $X_{ij}$. The evaluation function $f(P_i)$ is defined to minimize the function given by Equation (4.19) for a given load demand $P_D$, while satisfying the constraints.

$$f(P_i) = \sum_{i=1}^{N} F_i(P_i) + \lambda \cdot \left[ \sum_{i=1}^{N} P_i - P_D \right]^2 + \gamma \cdot \left[ \sum_{i=1}^{n} v_i^R \right]$$  \hspace{1cm} (4.19)

where $\lambda$ is the penalty parameter for not satisfying the load demand and $\gamma$ represents the penalty for a unit loading falling within a prohibited operating zone. $v_i^R$ is the violation of the prohibited zone constraint for the $i$th unit which is defined as

$$v_i^R = \begin{cases} 
1 & \text{if } P_i \text{ violates the prohibited zones} \\
0 & \text{} 
\end{cases}$$  \hspace{1cm} (4.20)

Step 4) Mutation operation

For each target vector $X_{i,G}$, a mutant vector $V_{i,G}$ is generated for the strategy DE/rand/1 using Equation (4.12).
Step 5) Recombination

Recombination is employed to generate a trial vector $U_i$ by replacing certain parameters of $X_i$ with corresponding parameters of donor vector $V_i$. The trial vector by crossover operation is obtained using Equation (4.12) and its fitness is evaluated using (4.14).

Step 6) Selection

Members to constitute the population of next generation (G+1) are decided by (3.15). The new vector $X_{i(G+1)}$ is selection based on the comparison of fitness value of both $X_i$ and $U_i$.

Step 7) Invoke VNS

After the selection process the VNS procedure is invoked as given in the Section 3.3.2.

Step 8) Verification of stopping criterion

Set the generation count $G=G+1$. Go to step 3 until stopping criterion is met usually maximum generation count $G_{max}$.

The flowchart for variable neighborhood search guided differential evolution is given in Fig 4.5.
Figure 4.5 Flowchart for variable neighborhood search guided differential evolution.

4.7 RESULTS AND DISCUSSION

In order to evaluate the performance of the proposed DE-VNS, the algorithm is coded in MATLAB language. This research suggested the values of parameters to be used in the DE-VNS are determined as follows:
1. Population size = 10,
2. Crossover probability = 0.8,
3. Mutation probability = 0.05,
4. Selection method = Tournament selection,
5. The maximum number of generations = 1000.

Random task graphs are generated with the task numbers ranging from 8 to 30. The number of processors varies from 2 to 10. The initial population is generated randomly with population size as 10. The tasks are scheduled to the processors by using the Genetic Algorithm. Table 4.1 shows the makespan for the DAGs of specified tasks by using three processors.

Table 4.1 Comparison of Basic Genetic Algorithm Makespan time and Proposed DE-VNS

<table>
<thead>
<tr>
<th>Task No</th>
<th>No of Tasks</th>
<th>No of Processors</th>
<th>Makespan time of Basic Genetic Algorithm (sec)</th>
<th>Makespan time of Proposed DE-VNS (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8</td>
<td>3</td>
<td>32</td>
<td>28</td>
</tr>
<tr>
<td>2</td>
<td>12</td>
<td>3</td>
<td>53</td>
<td>49</td>
</tr>
<tr>
<td>3</td>
<td>16</td>
<td>3</td>
<td>72</td>
<td>65</td>
</tr>
<tr>
<td>4</td>
<td>20</td>
<td>3</td>
<td>86</td>
<td>82</td>
</tr>
<tr>
<td>5</td>
<td>24</td>
<td>3</td>
<td>92</td>
<td>89</td>
</tr>
</tbody>
</table>

From the Table 4.1, it is obvious that the makespan obtained by using the new algorithm is less compared to the basic algorithm. In table 4.2, the execution time obtained for different DAGs using the basic GA and the proposed DE-VNS is compared. The number of processors used is 4. The execution time for the new algorithm is found to be less than the old algorithm.
Table 4.2  Comparison of Basic Genetic Algorithms and the proposed DE-VNS execution time

<table>
<thead>
<tr>
<th>Task No</th>
<th>No of Tasks</th>
<th>No of Processors</th>
<th>Execution Time by using Basic GA (in sec)</th>
<th>Execution Time By Proposed DE-VNS (in sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8</td>
<td>4</td>
<td>40</td>
<td>32</td>
</tr>
<tr>
<td>2</td>
<td>12</td>
<td>4</td>
<td>52</td>
<td>43</td>
</tr>
<tr>
<td>3</td>
<td>16</td>
<td>4</td>
<td>61</td>
<td>55</td>
</tr>
<tr>
<td>4</td>
<td>20</td>
<td>4</td>
<td>73</td>
<td>68</td>
</tr>
<tr>
<td>5</td>
<td>24</td>
<td>4</td>
<td>82</td>
<td>78</td>
</tr>
</tbody>
</table>

Figure 4.6 show the comparison of basic Genetic Algorithm and proposed hybrid Genetic Algorithm makespan time. From the Figure 4.2, it is clear that the makespan time of proposed method is less for scheduling 24 tasks compared to the basic Genetic Algorithm.

Makespan time is the total completion time of all the tasks. Figures 4.7 show the comparison of basic Genetic Algorithm and proposed hybrid Genetic Algorithm makespan time. It is clear that the makespan time of proposed method is 88 sec, whereas existing GA takes 92 sec. for scheduling 24 tasks. The performance of the proposed DE-VNS is 4% better than the existing GA less compared to the basic Genetic Algorithm.
Figures 4.6 and 4.7 show the comparison of basic Genetic Algorithm and proposed hybrid Genetic Algorithm execution time. It is clear that the execution
time of proposed method takes 88 sec. whereas existing GA takes 92 sec. for scheduling 24 tasks. The performance of the proposed DE-VNS is 4% better than the existing GA less compared to the basic Genetic Algorithm.

Additionally, we build a simulation and evaluation situation to assess the proposed method. The testing is carried out in four autonomous parts which will be implemented in sequence. The first part is a task graph generator. Based on the quantity of tasks the user participates for simulation, it will generate task graphs randomly.

This research adopts the *B-level Partition Algorithm* for implementation in the second part, which will partition complete task graph into a number of subgroups allocated by the user. The third part is the most critical and vested one as for the simulation is concerned. This stage applies the proposed Differential Evolution with Variable Neighborhood Search algorithm to schedule all subgroups in succession and produce local schedules. The fourth or the last segment is utilized to get the better of all local schedules and re-estimate the complete makespan for the scheduling in the given multiprocessor systems.

To organize the number of tasks in each subgroup, two sets of task graphs to estimate the proposed method are generated. The simulation experiments are arranged such that the first set contains 40 to 100 tasks in its task graphs. They are classified into 1~5 subgroups and has been executed on a multiprocessor with 4 processors. Similarly the task graphs in the next set include 100 to 500 tasks. To execute the second set of Tasks, we partition them into 1~10 subgroups and use a multiprocessor system with 8 processors. In the following, experimental results for both task graph sets are shown together. Meanwhile, when scheduling each subgroup, the number of populations is adapted to the number of tasks in that subgroup.
1. First set of Task graphs

2. Second set of Task graphs

**Figure 4.8: Performance results for the proposed algorithm**

Fig. 4.8, shows the performances of proposed Differential Evolution with Variable Neighborhood Search algorithm with different number of subgroups.
1. First set of Task graphs

2. Second set of Task graphs

Figure 4.9: Computational time (Normalized) for the proposed algorithm

These performances (makespans) are normalized, and Differential Evolution with Variable Neighborhood Search with only one subgroup is fundamentally the same as original existing algorithms. Analysing the results we can perceive that the normalized performances only vary between 0.963 and
1.025. It indicates that the scheduling ability of proposed Differential Evolution algorithm is similar as genetic algorithm, sometimes it is even better.

1. First set of Task graphs

2. Second set of Task graphs

Figure 4.10: Computational time (actual) for the proposed algorithm
In Fig.4.9, it is clear that proposed algorithm can considerably reduce the scheduling time. From the experimental simulations, the reduction of scheduling time is evident only for less number of subgroups. After that the time variation is very slight.

Finally, Fig.4.10, demonstrates that the scheduling time with various pairs of number of tasks and subgroups. It is obvious that if we partition the task graph into more subgroups, the scheduling time increases much slower when the number of tasks becomes larger. Relations between scheduling time and number of subgroups are not linear. This result indicates proposed Differential Evolution with Variable Neighborhood Search algorithm is scalable than original existing algorithm, which can ultimately extend its feasibility.

4.8 SUMMARY

The problem of scheduling of tasks to be executed on a multiprocessor system is one of the most challenging problems in parallel computing. In this research work, DE-VNS with a modified mutation operator for the multiprocessor scheduling problem is used to get a reasonable finishing time or makespan of the schedule. The tasks have precedence relationship and are represented by using Directed Acyclic Graph. The individuals are represented in the form of vectors having a number of cells. The precedence relations of the tasks are considered. The initial population is generated randomly and fitness values are calculated for each individual. The crossover and mutation operations are performed until algorithm converges. The crossover of the tasks also guarantees that the new strings generated are legal. Furthermore, a tabu list is used for avoiding repetition of the parents whose children have been already searched in previous generations. The proposed algorithm will give a reduced finish time or makespan for the schedule of tasks
represented by using the DAG with a reasonable execution time. The experimental result indicated that better solutions with less execution time are obtained by a combination of Genetic Algorithm.