Reinforcement Learning approaches to Economic Dispatch problem

E.A. Jasmin, T.P. Imthias Ahamed, V.P. Jagathy Raj

Abstract

This paper presents Reinforcement Learning (RL) approaches to Economic Dispatch problem. In this paper, formulation of Economic Dispatch as a multi stage decision making problem is carried out, then two variants of RL algorithms are presented. A third algorithm which takes into consideration the transmission losses is also explained. Efficiency and flexibility of the proposed algorithms are demonstrated through different representative systems: a three generator system with given generation cost table, IEEE 30 bus system with quadratic cost functions, 10 generator system having piecewise quadratic cost functions and a 20 generator system considering transmission losses. A comparison of the computation times of different algorithms is also carried out.

1. Introduction

Reinforcement Learning Problem in general is a problem of learning from interaction to achieve a specified goal. The learner (decision maker) continuously interacts with the environment (system). The interaction is through actions and associated rewards. The agent performs an action from the permissible set of actions at the particular state of the environment. The environment gives back a numerical reward which is a measure of the goodness of the performed action. Such a learning scheme is widely employed in solving several difficult problems such as control of inverted pendulum, Playing Backgammon and other computer games, Elevator control, etc. There are few applications of Reinforcement Learning in Power System problems also. It has been applied for Load Frequency Control of generators, Unit Commitment problem, Power system transient stability enhancement, Auction Based Pricing, Optimal bidding of a Genco.

As far as power generation control is considered, it is basically having three time based control loops: Unit Commitment, Economic Dispatch and Load Frequency Control. In Economic Dispatch problem, cost of generation of power has been represented in a variety forms including cost tables, quadratic functions, etc. For getting more accurate representation, cost functions are also sometimes expressed as piecewise quadratic functions. For solving this scheduling problem, so many computational and intelligent techniques have been developed so far. Some of the strategies applied for solution of this problem are explained in [14]. Fast computation Hopfield neural network along with dynamic programming is used for getting the schedule of generation in [15]. The work presented in [16] explains the application of Hopfield Neural Network for Economic and Emission Load Dispatch. The work presented in [17] also gives out the idea of using Hopfield Neural Network as a tool for solving this control problem. Particle swarm optimization is the technique used by [18] and [19] considers the security constraints of the problem also is solved through decomposition and coordination algorithms and variable scaling hybrid differential programming method respectively. [20] uses an immune based method known as Clonal algorithm. Using Radial Basis Function Network to compute optimum value for lambda and then using lambda iteration method the problem is solved in [21]. Simple Genetic algorithm is used for finding optimum dispatch [22] and simulated annealing is used as tool in [23]. Piece wise quadratic functions are considered and solution is made through an improved genetic algorithm in [24]. Recently a direct search method viewing the non convex nature of economic dispatch problem is employed in the work presented in [25] and [26] also gives out a direct search approach. Different kinds of efficient evolutionary algorithms are developed in [27] and [28]. Particle swarm optimization is the technique used by the researchers in [29] and [30] while a new optimization based algorithm: ‘Taguchi method’ is used in [31].

Reinforcement Learning seems to provide better flexibility and easiness in accommodating randomness in the cost strategies associated with complex systems. Also since this learning strategy relies on an evaluative feedback approach, like other intelligent techniques, it can work on systems with ill defined models. In

© 2011 Elsevier Ltd. All rights reserved.
addition, the evaluation of reward corresponding to an action which is the core part of this strategy need not be based on mathematical functions. It just needs a numerical value indicating the goodness of a particular schedule. In this approach, we treat Economic Dispatch as a multi stage decision making problem which distributes the power demand among the generating units available on line in such a way as to minimize operating cost. Recently it was shown that RL algorithms could be applied to solve Economic Dispatch problem by considering a simple three generator system having quadratic cost functions [11,12]. In this paper a class of RL algorithms are proposed. Performance analysis of the algorithms are carried out considering different test cases with different types of cost functions.

The organization of the rest of the paper is as follows. In Section 2, Economic Dispatch problem is presented in terms of mathematical equations. Section 3 gives an explanatory discussion on Reinforcement Learning and introduces the terms and notations used in RL algorithm using the example of shortest path problem. In Section 4 Economic Dispatch is formulated as a multi stage decision making problem. Section 5 explains the Reinforcement Learning algorithms for solution of this decision making problem. Transmission losses are also considered and algorithm for solving the same is presented in Section 6. Performance evaluation of the proposed solution methods using the different case studies are given in Section 7. Conclusion with discussion on future improvements possible follows.

2. Economic Dispatch

Economic Dispatch is basically to find the relative loadings of the various generating units online.

The allotment should be in such a way that the cost of generation should be minimum as far as possible. At the same time generating unit power constraints should also be met. Therefore the objective function of Economic Dispatch problem is equal to the total cost for supplying the demanded load $P_T$. The problem is to minimize $F_T$ subject to the constraints that the total generated power and the demanded load equals and the power constraints on all units are being met. Mathematically the objective function $F_T$ can be expressed as

$$F_T = F_1(P_1) + F_2(P_2) + F_3(P_3) + \ldots + F_N(P_N) = \sum_{i=1}^{N} F_i(P_i)$$

where $F_i$ denotes the cost function of $i$th unit and $P_i$ the electrical power generated by that particular unit and constraints are

$$P_T = \sum_{i=1}^{N} P_i = 0$$

$$P_{\text{min i}} < P_i < P_{\text{max i}} \quad \text{for} \quad i = 1 \text{ to } N$$

where $P_T$, total load power demand; $P_{\text{min i}}$, min. power generation of $i$th unit; $P_{\text{max i}}$, max. power generation of $i$th unit.

Therefore the problem is to minimize the cost function $F_T$ subject to the constraints given in Eqs. (2) and (3).

3. Reinforcement Learning

Reinforcement Learning is one effective method in the solution of multi stage decision making problems. For a comprehensive study of the subject, refer [1,2,32,33]. To make the paper self contained, a brief introduction is given here.

A general layout of Reinforcement Learning task is given in Fig. 1. The agent interacts with the environment through actions and associated rewards. It uses training information that evaluates actions (in terms of reward received from the environment on performing an action) taken by the agent. The effect of action at any instant depends on nature of the problem environment.

In order to make the concepts clear and to explain the steps followed in the learning task let us consider the example of a simple shortest path problem. Consider a grid world problem shown in Fig. 2.

The grid considered is having 36 cells arranged in 6 rows and 6 columns. A robot can be at any one of the possible cells at any instant. We can refer the cell number as state of the robot. $G$ denotes the goal state to which the robot aim to reach and the crossed cells denote cells with some sort of obstacles. There is cost associated with each cell transition while the cost of passing through a cell with obstacle is much higher compared to other cells. Starting from any initial position in the grid, robot can reach the goal cell by following different paths and correspondingly cost incurred will also vary. The problem is to find an optimum path to reach the goal state (cell) starting from any one of the initial state. We can denote the state of the robot at instant $k$ as $x_k$. At any instant $k$, the robot can take any of the action (movement) $a_k$ from the set of permissible actions in the action set $A_k$ which also depend on the current state $x_k$. For example if $x_k = 7$, $A_k = \{\text{right, up, down}\}$ and if $x_k = 1$, $A_k = \{\text{right, down}\}$. The state occupied by the robot in $k+1$, $x_{k+1}$ depends on $x_k$ and $a_k$. That is,

$$x_{k+1} = f(x_k, a_k) \quad (4)$$

For example, if $x_k = 7$ and $a_k = \text{Down}$ then $x_{k+1} = 13$ while when $a_k = \text{Up}$, $x_{k+1} = 1$. The state $x_{k+1}$ can be found from the simulation model of the grid or studying the environment in which robot moves.

Therefore the shortest path problem can be stated as finding the sequence of actions $a_0, a_1, \ldots, a_{T-1}$ starting from any initial state such that the total cost for reaching goal state $G$ is minimum. The numerical cost when the agent (robot) performs an action $a_k$ at state $x_k$ making transition to $x_{k+1}$ is denoted as $r_k$, known as immediate reward or reinforcement. This reward $r_k$ in general can depend on current state, action and the next state. That is,

$$r_k = G(x_k, a_k, x_{k+1}) \quad (5)$$

![Fig. 1. A general layout of Reinforcement Learning task.](image1)

![Fig. 2. Grid world problem.](image2)
In this simple grid world we assume a cost of 1 unit for transition to an ordinary cell while 10 units for a cell having some obstacles.

\[ g(x_k, a_k, x_{k+1}) = 1, \quad \text{if } x_{k+1} \text{ is a normal cell} \]

\[ = 10, \quad \text{if } x_{k+1} \text{ is a cell with obstacles} \]

To find the total cost, the cost on each transition can be cumulated. Now the total cost for reaching the goal state can be taken as

\[ \sum_{T=1}^{T} g(x_k, a_k, x_{k+1}), \quad x_0 \text{ being the initial state and } T \text{ being the no. of transitions to reach the goal state}. \]

The action selection can be considered as following a policy \( \pi \), i.e., \( \pi(x) \) denotes the action taken on reaching the state \( x \). The Reinforcement Learning task now reduces to finding an optimum policy \( \pi^* \) which gives out the optimum action at each state (cell) in order to get minimum total cost. For this problem, the aim is to minimize the total cost which is equal to \( \sum_{T=1}^{T=1} g(x_k, a_k, x_{k+1}) \).

In general, this need not be the case. Here we are assuming cost incurred at all stages have equal importance. However in some cases a cost of 10$ incurred at a later stage may not have the same effect as a cost of 10$ at the current stage. Hence future cost may be discounted by a factor. In such situations, total cost is defined as,

\[ \text{cost} = \sum_{k=0}^{T-1} r_k g(x_k, a_k, x_{k+1}) \]

In more a general setting, cost incurred may be a random variable. In such a problem the objective function is \( E\{\sum_{k=0}^{T-1} g(x_k, a_k, x_{k+1})\} \) which is the expected cost.

Now we introduce \( Q \) value. \( Q \) value of a multi stage problem is defined as:

\[ Q^k(x, a) = E^{k+1} \sum_{k=0}^{T} g(x_k, a_k, x_{k+1}) \]

\( Q^k(x, a) \) is the expected long term reinforcement when we start in state (cell) \( x \), take action \( a \) in the starting state and thereafter follow the policy \( \pi \) and \( \gamma \) is the discount factor. For example \( Q^k(2, D) \) denote the \( Q \) value associated with the cell 2 and starting with action Down. Similarly each of the different cells and possible actions will be associated with a particular \( Q \) value. Now define \( Q^k(x_0, a_0) \) as the expected value of the total cost incurred in taking an action \( a_0 \) from the state \( x_0 \) and thereafter taking actions based on policy \( \pi \). One can call \( Q^k(x_0, a_0) \) as an optimal \( Q \) function if \( Q^k(x_0, a_0) = \min_{a} Q^k(x_0, a) \). Therefore, now the optimal policy can be stated as: \( \pi^* (x_k) = \arg \min_a Q^k(x_k, a) \), \( a \in A_k \), \( x_k \) being the action to be taken at state \( x_k \) in order to get minimum cost.

Therefore if one can find \( Q^k(x_0, a_0) \) for all state–action pairs, then the path for reaching the goal state can be traced out from any initial state at minimum cost. In Q learning algorithm, first all \( Q \) values are initialized with some initial value, \( Q^0(x_0, a_0) \). At each iteration \( n \), on reaching \( x_n \), an action \( a_n \) is taken based on the current estimate of \( Q^k(x_n, a_0) \), i.e., \( Q^k(x_n, a_0) \). Once action is taken at state \( x_n \), it makes transition to \( x_{n+1} \) and the reward \( g(x_n, a_n, x_{n+1}) \) can be found from simulation. The \( Q \) value is updated using the equation,

\[ Q^{n+1}(x_n, a_n) = Q^n(x_n, a_n) + \alpha [g(x_n, a_n, x_{n+1}) + \gamma \arg \min_{a'} Q^n(x_{n+1}, a') - Q^n(x_n, a_n)] \]

\[ 0 < \alpha < 1 \] is a constant and is called step size of learning.

One issue to be settled is how to choose action in each step. At the nth iteration, \( Q \) is unknown but \( Q^n \) (estimate of \( Q \)) is known to the learning agent. If \( Q^n \) is same as \( Q \), the best action is \( a_g = \arg \min_{a} Q^n(x_n, a) \). This way of choosing action is called greedy algorithm and \( a_g \) is called greedy action. If one chooses greedy action initially, it may be wrong if \( Q^n \) is not equal to \( Q \).

One choice is to go for \( \epsilon \) – greedy algorithm. Here Greedy action \( a_g \) will be chosen with a probability of \( (1 - \epsilon) \) and one among all other permissible actions is chosen with a probability of \( \epsilon \). \( \epsilon \) can be taken as fixed value throughout the learning. But a small fixed value may slow down the learning process, while a large fixed value may make the learning process not to converge. To overcome these problems, an adaptive method is used for fixing of exploration rate. In this approach \( \epsilon \) is fixed arbitrarily at some sufficiently large value and then progressively reduced to make a smooth and fast convergence possible.

The \( \epsilon \) – greedy method discussed above provides a good method of action selection, for providing better exploration in the initial phases of learning while exploiting the goodness of greedy action during the later phases. However \( \epsilon \) – greedy requires a gradual reduction of \( \epsilon \). That is, a proper cooling schedule is to be designed which gradually updates the value of \( \epsilon \) as the learning proceeds so that proper convergence and correctness of the result are assured. The length of learning phase mainly depends on this cooling schedule and therefore it is one significant part of \( \epsilon \) – greedy method. It is a difficult task to develop a good cooling schedule so as to ensure minimum time for convergence.

Another stochastic policy followed for selection of action in the Reinforcement Learning task is Pursuit algorithm. In this method along with maintaining estimates of \( Q \) values as measure of goodness of actions, some preference is also associated with actions. Each action \( a_k \) at any state \( x_k \) is having a probability \( p_{a_k}(a_k) \) of being chosen. These probability values will be same for all actions and all states initially assuring sufficient exploration of the action space. Then on performing an action \( a_k \) at any state \( x_k \) during learning, the numerical reward is used to update the estimate of \( Q \) value associated with the state–action pair. Along with that, based on the current estimates of \( Q \) values, probability values associated with actions are also modified as:

\[ p_{a_k}(a_k) = p_{a_k}(a_k) \]

\[ p_{a_k}(a_k) = p_{a_k}(a_k) - \beta p_{a_k}(a_k), \quad \text{when } a_k = a_g \]

\[ p_{a_k}(a_k) = \left[ p_{a_k}(a_k) \right] - \beta p_{a_k}(a_k), \quad \text{when } a_k \neq a_g \]

where \( 0 < \beta < 1 \) is a constant. Thus at each iteration \( n \) of the learning phase, algorithm will slightly increase the probability of choosing the greedy action \( a_g \) in state \( x_k \) and proportionally decrease the probability associated with all other possible actions. Initially since all probabilities are made equal, sufficient exploration of action space is assured. When the algorithm proceeds a number of iterations, with high probability \( Q^n(x, a) \) will approach to \( Q^n(x, a) \) corresponding to all states. This is because, when the parameter \( \beta \) is properly chosen, after sufficient number of iterations, the greedy action in state \( x \), with respect to \( Q^n \) would be the same as greedy action in state \( x \) with respect to \( Q \) which gives the optimal action. In other words, through the iterative updating of probabilities, probability of optimal action increases successively. This in turn indicates an increase in probability of selecting the optimum action in the succeeding steps. If \( \alpha \) and \( \beta \) are sufficiently small, \( Q^n(x, a) \) would converge with high probability to unity.

In both these methods of solution, the problem is made to start from any of the initial states at random at each of the iteration of learning so that it goes through different transition paths updating the corresponding \( Q \) values [1]. In the next sections, solution to Economic Dispatch is obtained using the above two methods for exploration of action space.

4. Economic Dispatch as multi stage decision problem

To view Economic Dispatch as a multi stage decision making problem, the various stages of the problem are to be identified. Consider a system with \( N \) generating units committed for dispatch. Then Economic Dispatch problem involves deciding the amount of power to be dispatched by \( G_0, G_1, G_2, \ldots \ldots \ldots \ldots, G_{N-1} \).
In this formulation the amount of power to be dispatched by \( G_k \) is denoted as action \( a_k \). Action \( a_k \) in Reinforcement Learning terminology, corresponds to a power allocation \( P_k \text{MW} \) to generating unit \( G_k \). \( P_k \) is numerically same as \( a_k \). Therefore, the action set \( A_k \) consists of the different values of power dispatch possible to \( G_k \). That is, \( A_k = \{ \text{Min}_{i} , \ldots , \ldots , \text{Max}_{i} \} \), \( \text{Min}_{i} \) being the minimum value of power that can be allotted to \( G_k \) and \( \text{Max}_{i} \) being the maximum power that can be allotted to \( G_k \). Values of \( \text{Min}_{i} \) and \( \text{Max}_{i} \) depend on the minimum and maximum values of power generation possible with the \( k \)th unit and also maximum and minimum power that can be allotted among the remaining \( N-k \) units available. Therefore, action set \( A_k \) is a dynamically varying one depending on the power already allotted to the previously considered units.

The quantization step (in MW power) is chosen based on the accuracy needed. A very small value is not necessitated due to the setting of the reference point setting in a plant. An optimum value is chosen based on the accuracy needed and the setting of the units. Also as number of generating units and hence the range of possible demand increases, the number of states in the state space increases. State space is also discretized to have definite units. Also as number of generating units and hence the range of possible power increases, the part of state space to be considered \((N-k)\) generating units. At each step, the Economic Dispatch algorithm will select an action corresponding to each state, Reinforcement Learning based solutions are explored. Solutions consist of two phases: learning phase and policy retrieval phase.

One major issue in the learning phase is how to develop an action selection strategy which can balance exploitation of available information and exploration of action space to learn new possibilities. There are various algorithms in RL literature \([1,2,32,33]\) which balances exploration and exploitation. Here we develop RL algorithms for economic dispatch using two action selection strategies.

In algorithm I, a conceptually simple \( \varepsilon - \) greedy algorithm is used for action selection. In algorithm II Pursuit algorithm is used. In the next two section these learning algorithms are presented.

5. RL algorithms for Economic Dispatch

In the previous section, Economic Dispatch is formulated as a multi stage decision making problem. To find the best action corresponding to each state, Reinforcement Learning based solutions are explored. Solutions consist of two phases: learning phase and policy retrieval phase.

5.1. Algorithm I

For solving this multi stage problem using Reinforcement Learning, first step is fixing of state space \( \chi \), action space \( A \) and the immediate reinforcement function \( g(x_k, a_k, x_{k+1}) \) precisely. The different units can be considered arbitrarily corresponding to the different stages.

Fixing of state space \( \chi \) primarily depends on number of generating units available on line and the possible values of power demand (which in turn directly depends on min. and max. values of power generation possible with each unit). Since there are \( N \) stages for solution of the problem, the state space is also divided into \( N \) subspaces. Thus, if there are \( N \) units to be dispatched, \( \chi = \chi_1 \cup \chi_2 \cup \ldots \ldots \cup \chi_{N-1} \).

The dispatch problem should go through \( N \) (no: of generating units) stages for making allocation to each of the \( N \) generating units. At any stage \((\text{stage}_k)\), the part of state space to be considered \((\chi_k)\) consists of the different tuples having the stage number as \( k \) and power values varying from \( \text{D}_{\text{min}(k)} \) to \( \text{D}_{\text{max}(k)} \) being the minimum power possible to be met with \( N-k \) units and \( \text{D}_{\text{max}(k)} \) the maximum power possible with \( N-k \) units.

That is, \( \chi_k = \{(k, \text{D}_{\text{min}(k)}) , \ldots , (k, \text{D}_{\text{max}(k)})\} \), where \( \text{D}_{\text{min}(k)} = \) minimum power possible with \( N-k \) units

\[
= \sum_{i=k}^{i=N-1} P_{\text{min}(i)}
\]

\( \text{D}_{\text{max}(k)} = \) maximum power possible with \( N-k \) units

\[
= \sum_{i=k}^{i=N-1} P_{\text{max}(i)}
\]

At each step, the Economic Dispatch algorithm will select an action from the permissible set of discretised values and forward the system to one among the next permissible states.

The action set \( A_k \) consists of different values of MW power that can be allotted to \( k^{th} \) unit. The action set \( A_k \) depends on the demand value \( D_k \) at the current state \( x_k \) and also on the minimum and
maximum power generation possible with remaining N−k units. Therefore action set A_k is dynamic in nature in the sense that it depends on the power already allotted up to that stage and also the minimum and maximum generation possible with the remaining N−k units. If D_k is the power to be allotted, minimum value and maximum value of action a_k are defined as

\[
\begin{align*}
\min_k &= \max[(D_k - D_{\text{max}(k)}), P_{\text{min}(k)}] \\
\max_k &= \min[(D_k - D_{\text{min}(k)}) / C_0, P_{\text{max}(k)}] \tag{7}
\end{align*}
\]

The number of elements in these sets \( \chi \) and \( A \) depends on the minimum and maximum limits and also the sampling step.

In Economic Dispatch problem, the reward function \( (g) \) can be chosen as the cost function itself. That is, reward received or cost incurred on taking an action a_k or allocating a power \( P_k \) at kth stage is the cost of generation of the power \( P_k \). In the Reinforcement Learning terminology, the immediate reward,

\[
r_k = g(x_k, a_k, x_{k+1}) = C_k(P_k) \tag{8}
\]

Initially Q values of all state–action pairs are set to zero. In each iteration greedy action \( a_k = \arg \min_{a_k} \{Q_n(x, a)\} \) is found. Then greedy action \( a_k \) will be chosen with a probability of \((1 - \varepsilon)\) and one among all other permissible actions is chosen with a probability of \( \varepsilon \). As explained earlier based on the action selected system moves to the next state. Since the aim is to minimize the cost of generation estimated Q values of state–action pair are modified at each step of learning as,

\[
Q^{n+1}(x_k, a_k) = Q^n(x_k, a_k) + \alpha \left[ g(x_k, a_k, x_{k+1}) + \gamma \min_{a_k} Q^n(x_{k+1}, a) - Q^n(x_k, a_k) \right] \tag{9}
\]

Here, \( \alpha \) is the step size of the learning algorithm and \( \gamma \) is the discount factor.

When the system comes to the last stage of decision making, there is no need of accounting the future effects and then the estimate of Q value is updated using the equation,

\[
Q^{n+1}(x_k, a_k) = Q^n(x_k, a_k) + \alpha g(x_k, a_k, x_{k+1}) - Q^n(x_k, a_k) \tag{10}
\]

As the learning steps are carried out sufficient number of times, estimated Q values of state–action pairs will approach to optimum so that we can easily retrieve the optimum policy (allocation) \( \pi^*(x) \) corresponding to any state \( x \).

The learning procedure can now be summarized. At each iteration of learning phase the algorithm will take the system initial condition (i.e., for \( k = 0 \)) which is the power demand, as one random value within permissible limits. Then an action is performed which will allocate power to one of the units and then pass to the next stage (\( k = 1 \)) with the remaining power. This proceeds until all the \( N - 1 \) units are allotted power. At each state transition step, the estimated Q value of the state–action pair is updated using Eq. (9).

As the learning reaches the last stage, since there is no choice of action, the remaining power to be allotted will be the power corresponding to the action \( \{a_{N-1} = D_{N-1}\} \). Then the Q value is updated using Eq. (10). The transition process is repeated a number of times (iterations) with random values of initial demand and each time the dispatch process goes through all the \( N - 1 \) stages. Value of \( \varepsilon \) is taken closer to 0.5 in the initial phases of learning and is reduced in every max_iter/10 iterations by 0.04.

The entire algorithm of learning using \( \varepsilon \) greedy is given in Table 1:

5.2. Algorithm II

In this algorithm, we use pursuit for action selection in each step. In case of Pursuit algorithm, for any given state \( x_k \), an action is selected based on the probability distribution function \( p_{a_k} \). In the case of Economic Dispatch, initially the probability associated with each action \( a_k \) in the action set \( A \) corresponding to \( x_k \) are initialized with equal values as

\[
p_{a_k}(a_k) = \frac{1}{n_k}
\]

where \( n_k \) is total number of permissible actions at stage \( k \). As in the previous algorithm, initialize the Q values of all state–action pairs to zero.

Then at each iteration step, an action \( a_k \) is selected based on the probability distribution. On performing action \( a_k \), it reaches the next stage with \( D_{k+1} = D_k - a_k \). The cost incurred in each step of learning is calculated as the sum of cost of producing power \( P_k \) with the kth generating unit. Q values are then updated using Eq. (9). At each of the iteration of learning, we find the greedy action as \( a_g = \arg \min_{a_k} \{Q_n(x, a)\} \). Then accordingly the probabilities of actions in the action set are also updated as,

\[
p_{a_k}^{n+1}(a_k) = p_{a_k}^{n}(a_k) + \beta [1 - p_{a_k}^{n}(a_k)], \quad \text{when } a_k = a_g \\
p_{a_k}^{n+1}(a_k) = p_{a_k}^{n}(a_k) - \beta p_{a_k}^{n}(a_k)], \quad \text{when } a_k \neq a_g \tag{11}
\]

The algorithm proceeds through several iterations until the probability of best action in each hour is increased sufficiently which indicate convergence of the algorithm. The entire algorithm is given in Table 2.

5.3. Policy retrieval algorithm

As the learning proceeds and updating of Q values of state–action pairs are done sufficiently large number of times, \( Q^n \) will be almost equal to \( Q \). Then the learned Q values are used to obtain the optimum dispatch. For any value of power demand \( P_D \), initialize \( D_0 = P_D \). Then the state of the system is \( (0, D_0) \). Find the greedy action \( a_g \) at this stage which is the best allocation for 0th generating unit \( P_0 \). The learning system reaches the next state as \( (1, D_1) \) where \( D_1 = D_0 - a_g \) find the greedy action corresponding to stage, as \( a_1 \). This proceeds up to \( (N-1) \)th stage. Then a set of actions (allocations) \( a_0, a_1, a_2, \ldots, a_{N-1} \) is obtained which is the optimum schedule \( P_0, P_1, \ldots, P_{N-1} \) of generation corresponding to power \( P_D \). The algorithm for getting the schedule from the learnt system follows:
The loss in a transmission network can be estimated by executing power flow algorithm or can be approximated using B-matrix loss formula. In order to find the schedule accounting the transmission losses, one of the previous algorithms can be used to carry out the learning steps. Being more efficient and faster, pursuit learning strategy is employed for learning the system, generating random possible values of load demand is presented in Table 3.

Thus, on executing the learning algorithm and then retrieving the schedule by finding the greedy action corresponding to the input power to each stage of the multi stage decision making task, optimum schedule is obtained for any value of load demand.

Till now, the transmission losses in the system are neglected. Now the Reinforcement Learning approach is extended in order to accommodate the transmission losses occurring in the system.

### 6. RL algorithm for Economic Dispatch considering transmission losses

The loss in a transmission network can be estimated by executing power flow algorithm or can be approximated using B-matrix loss formula. In order to find the schedule accounting the transmission losses, one of the previous algorithms can be used to carry out the learning steps. Being more efficient and faster, pursuit learning strategy is employed for learning the system, generating random values of initial demand. Once the learning phase is completed, the policy retrieval steps provide us with the optimum schedule for any load value. In order to incorporate the transmission losses, the learning is carried out first and policy retrieval is done successively for different values of load values, accounting the losses.

First learn the Q value for the different state–action pairs. Schedule for the required load demand is retrieved by policy retrieval phase. For the schedule obtained, transmission losses are calculated by either finding the power flows or using B-matrix loss formula. The input demand is then modified by adding the calculated loss MW. The learning algorithm proceeds to find the dispatch for the new demand value, giving out the new schedule of generation. This new power allocation will certainly give a new value of transmission loss, which is again used for updating the demand.

The iterative procedure is continued until the loss calculation converges (indicated by the change in loss in two successive iterations coming within tolerable limits). By following these steps iteratively for different load values ranging from \( D_{\text{min}}(0) \) to \( D_{\text{max}}(0) \), economic allocation schedule for the entire range of possible demand (with the given generating units) is obtained. Algorithm incorporating transmission loss to find the schedule for all the possible values of load demand is presented in Table 3.

Thus the discrete step for load MW is taken as 10 MW so as to manage the number of states at each stage of the problem. Value of \( \mu \) is taken as 1 MW so that transmission loss, less than 1 MW can be neglected compared to the load power. Once the learning phase is completed, the economic allocation for all the possible load values can be obtained instantaneously. The main attraction of these algorithms comes from the fact that the learning is carried out only once and need not be repeated for each load demand as in other stochastic methods.

### 7. Performance of algorithms

The proposed Economic Dispatch algorithms are assessed using different standard test cases. RL based Economic Dispatch can be applied for finding the schedule for generating units when the cost of generation is provided in any of the different forms like variable cost table, cost coefficient, piecewise cost functions and actual cost data from a plant. This becomes useful when the cost of power varies in every block of time since the availability of power is practically a dynamic one.

Algorithms are coded in C language and compiled and executed in GNU Linux environment. Performance evaluation is done with Pentium IV, 2.9 GHz, 512MB RAM personal computer.

In order to validate the proposed algorithms and make a comparison among them, first a three generator system with cost data given in a tabular form is considered\[13\]. The first two algorithms are executed to find the dispatch without transmission losses and a comparison of execution time is made.

Then IEEE 30 bus system with six generating units having quadratic cost functions \[34\] is taken in order to prove the efficacy of the proposed approaches. The suitability of the proposed algorithms for a system having generating units with piecewise cost functions is also studied by considering a 10 generator system. The last algorithm is validated and the flexibility of the Reinforcement Learning solution is investigated for system with 20 generating units having given cost functions.
In order to apply Reinforcement Learning algorithms, first the learning parameters are to be fixed based on the problem environment. The learning parameter $\epsilon$ accounts for rate of exploration and exploitation needed. Since it indicates a probability, it can take any positive value less than 1. A small fixed value may result in premature convergence of the learning algorithm while a large fixed value may make the system oscillatory. Therefore in these RL based algorithms, a value of 0.5 is assumed initially providing sufficient exploration of the search space and is decreased by a small factor successively.

Discount parameter $\gamma$ accounts for the discount to be made in the present state for accounting of future reinforcements and since in the case of Economic Dispatch problem, the cost of future stages has the same implication as the cost of the current stage, value of $\gamma$ is taken as 1.

The step size of learning is given by the parameter $\alpha$ and it affects the rate of modification of the estimate of $Q$ value at each iteration step. By trial and error $\alpha$ is taken as 0.1 for achieving sufficiently good convergence of the learning system. The RL parameters used in the dispatch problem are tabulated in Table 4.

### Case I – three generator system

First a simple example with three generating units [13] is considered for validating and explaining the RL approach of solution. The transmission losses are neglected in this case. The cost details are given in tabular form, which can be obtained from experience in case of a practical system. The unit characteristics are given in Table 5, where $C_i$ stands for the cost of generating $P$ MW by $i$th unit.

The three generating units are having the minimum and maximum power generation possible as (50, 200), (50, 150) and (50, 175). The discretization step for state space and action space, $S_i$ and $A_i$ are taken as 25 MW.

Therefore,$$
D_{min}(0) = 150, \quad D_{max}(0) = 525 \\
D_{min}(1) = 100, \quad D_{max}(1) = 325 \\
and D_{min}(2) = 50, \quad D_{max}(2) = 175
$$

In order to understand the performance of RL algorithms, the different components of the multi stage decision process are to be identified. The state tuples is of the form $(k, D_k)$, $D_k$ being the power to be dispatched at $k$th stage.

<table>
<thead>
<tr>
<th>Table 4</th>
<th>RL parameters.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\epsilon$</td>
<td>0.5</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>0.1</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>1</td>
</tr>
</tbody>
</table>

### Case II – IEEE 30 bus system

To prove the flexibility, the proposed algorithms are now tested for IEEE 30 bus system consisting of six generators [34], without considering the transmission losses. The system cost data is given in quadratic cost coefficient form as given in Table 8, i.e., for any

**Table 5** Cost table for three generator system.

<table>
<thead>
<tr>
<th>$P$ (MW)</th>
<th>$C_1$</th>
<th>$C_2$</th>
<th>$C_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>100,000</td>
<td>100,000</td>
<td>100,000</td>
</tr>
<tr>
<td>25</td>
<td>100,000</td>
<td>100,000</td>
<td>100,000</td>
</tr>
<tr>
<td>50</td>
<td>810</td>
<td>750</td>
<td>806</td>
</tr>
<tr>
<td>75</td>
<td>1355</td>
<td>1155</td>
<td>1108.5</td>
</tr>
<tr>
<td>100</td>
<td>1460</td>
<td>1360</td>
<td>1411</td>
</tr>
<tr>
<td>125</td>
<td>1772.5</td>
<td>1655</td>
<td>1710.5</td>
</tr>
<tr>
<td>150</td>
<td>2085</td>
<td>1950</td>
<td>1998</td>
</tr>
<tr>
<td>175</td>
<td>2427.5</td>
<td>100,000</td>
<td>2358</td>
</tr>
<tr>
<td>200</td>
<td>2760</td>
<td>100,000</td>
<td>10000</td>
</tr>
<tr>
<td>225</td>
<td>100,000</td>
<td>100,000</td>
<td>10000</td>
</tr>
</tbody>
</table>

**Table 6** Allocation schedule for three generator system.

<table>
<thead>
<tr>
<th>$D$ (MW)</th>
<th>$P_{1}$ (MW)</th>
<th>$P_{2}$ (MW)</th>
<th>$P_{3}$ (MW)</th>
<th>Cost (Rs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>250</td>
<td>50</td>
<td>50</td>
<td>150</td>
<td>3558</td>
</tr>
<tr>
<td>275</td>
<td>50</td>
<td>150</td>
<td>75</td>
<td>3868.5</td>
</tr>
<tr>
<td>300</td>
<td>50</td>
<td>100</td>
<td>150</td>
<td>4168</td>
</tr>
<tr>
<td>325</td>
<td>50</td>
<td>125</td>
<td>150</td>
<td>4463</td>
</tr>
<tr>
<td>350</td>
<td>50</td>
<td>150</td>
<td>150</td>
<td>4758</td>
</tr>
<tr>
<td>375</td>
<td>100</td>
<td>125</td>
<td>150</td>
<td>5113</td>
</tr>
<tr>
<td>400</td>
<td>100</td>
<td>150</td>
<td>150</td>
<td>5408</td>
</tr>
<tr>
<td>425</td>
<td>125</td>
<td>150</td>
<td>150</td>
<td>5720.5</td>
</tr>
<tr>
<td>450</td>
<td>150</td>
<td>150</td>
<td>150</td>
<td>6033</td>
</tr>
<tr>
<td>475</td>
<td>175</td>
<td>150</td>
<td>150</td>
<td>6373.5</td>
</tr>
<tr>
<td>500</td>
<td>200</td>
<td>150</td>
<td>150</td>
<td>6708</td>
</tr>
</tbody>
</table>

**Table 7** Comparison execution time for $\epsilon$ greedy and pursuit solutions.

<table>
<thead>
<tr>
<th></th>
<th>$\epsilon$ greedy</th>
<th>Pursuit</th>
</tr>
</thead>
<tbody>
<tr>
<td>No: of iterations</td>
<td>100,000</td>
<td>50,000</td>
</tr>
<tr>
<td>Computation time (s)</td>
<td>2.567</td>
<td>1.754</td>
</tr>
</tbody>
</table>

Then, state space $X = X_0U_1U_2U_3$ where

$X_0 = \{(0, 150), (0.175), (0.200), \ldots, (0.525)\}$

$X_1 = \{(1.100), (1.125), (1.150), \ldots, (1.325)\}$

and $X_2 = \{(2.50), (2.75), (2.100), \ldots, (2.175)\}$

Now identify the action space, which is a dynamic one since it depends on the value of power $D_1$ to be dispatched. The minimum and maximum values of actions are found out as

$Min_0 = (D_0 - D_{min(1)})$ or $P_{min(0)}$ whichever is greater

$Max_0 = (D_0 - D_{max(1)})$ or $P_{max(0)}$ whichever is smaller

$Min_1 = (D_1 - D_{max(1)})$ or $P_{max(1)}$ whichever is greater

$Max_1 = (D_1 - D_{min(1)})$ or $P_{max(1)}$ whichever is smaller

$Min_2 = P_{min(2)}$, $Max_2 = P_{max(2)}$

For the purpose of explaining the algorithm, let the random value generated for the demand be 300 MW. Then the action space at stage $0$ is

$A_0 = \{50, 75, \ldots, 200\}$

One of these actions is selected and it passes to the next stage $k = 1$. Then the action space of $A_1$ is identified and action selection is continued.

The two algorithms, based on $\epsilon$ greedy and pursuit are executed. Both the algorithm gave same results. The allocation schedule for a load demand of 300 is obtained as (50,100,150) and the cost of generation is Rs. 4168/- using policy retrieval phase. Power schedule for any value of possible input demand values can be retrieved. The two algorithms are run for various values of power demand $D_{min(0)} \leq D_0 \leq D_{max(0)}$, i.e., 150 $\leq D_0 \leq 525$. Part of the simulation result is tabulated in Table 6 which is consistent with results given in [13].

For comparing the efficacy of the two algorithms, Simulation time for the two algorithms are compared in Table 7.

On comparing the computation time and performance of the algorithms, Algorithm II seems to be better.

**Case II – IEEE 30 bus system**

To prove the flexibility, the proposed algorithms are now tested for IEEE 30 bus system consisting of six generators [34], without considering the transmission losses. The system cost data is given in quadratic cost coefficient form as given in Table 8, i.e., for any
power $P$, cost of generation is given out by the equation $C(P) = Ca + CbP + CcP^2$, where $Ca$, $Cb$ and $Cc$ are the cost coefficients. Also the maximum and minimum generations possilbe for each of the six generators are specified.

The maximum generation possible with these six generators turns out to be 2330 MW while minimum generation is 540 MW. The RL algorithms are now applied to get the economic allocation for the six units. The discretization step for action and state space are taken as 10 MW as balance between the accuracy and size of the state and action spaces. At each step of iteration, action is selected according to the exploration method. The Q values of state–action pairs are updated for which cost of generation is calculated by evaluating the quadratic equation.

In case of greedy method, after $5 \times 10^5$ iterations the Q values approach optimum while pursuit solution converged in $2 \times 10^5$ iterations. The optimum dispatch is found out by tracing out the greedy action which gives out minimum iterations. The optimum dispatch is found out by tracing out the power allocation schedule for IEEE 30 bus system.

### Case III – 10 Generator system with piece wise quadratic cost functions

To verify the algorithms for non convex cost functions and compare with one of the recent techniques, 10 generator system having piecewise quadratic cost functions [24] is considered. The different generators have two or three different operating regions. If the Cost function is $C_i$ and the space interval is divided into three divisions, then it is represented as follows:

$$
C_i(P) = a_{i1} + b_{i1}P + c_{i1}P^2 \quad (P_{\text{min}} \leq P_i \leq P_{\text{mid}})
$$

$$
= a_{i2} + b_{i2}P + c_{i2}P^2 \quad (P_{\text{mid}} < P_i \leq P_{\text{max}})
$$

$$
= a_{i3} + b_{i3}P + c_{i3}P^2 \quad (P_{\text{max}} < P_i)
$$

The data $(a_i, b_i, c_i, P_{\text{min}}, P_{\text{max}})$ of generators are given in Table 10.

The system is made to learn using the two algorithms given in Tables 1 and 2 and the Q values approach optimum in $5 \times 10^5$ and $1.5 \times 10^7$ iterations respectively. The same values of learning parameters are taken as in previous cases. The discretization step for state and action spaces is taken as 10 MW.

Part of the allocation schedule corresponding to values of power demand ranging from 1400 MW to 3000 MW obtained are given in Table 11. The times of execution are 38.69 s and 34.95 s respectively. The cost and allocation schedule obtained are comparable with that of improved genetic algorithm [24].

The simulation results proved the suitability and efficiency of the proposed algorithms for scheduling of generators with different categories of cost functions. Now incorporating the transmission losses, schedule for 20 generating unit system is obtained.

### Case IV – 20 Generator system

For validating the Reinforcement Learning algorithm accounting the transmission losses occurring in the system and to prove the scalability of Reinforcement Learning based algorithms, next a 20 generator system is considered. The cost function is given in quadratic form. The unit details are given in Table 12.

The transmission loss is calculated using $\beta$ coefficient matrix.

Algorithm given in Table 3 is executed to give the schedule for the range of load from 1000 MW to 3000 MW. The execution time taken is 37.97 s Part of the schedule and the loss are tabulated in Table 13.
The different Reinforcement Learning algorithms have been tested for their efficacy and performance. The computation time of the different RL algorithms for the different test cases are tabulated for comparison in Table 14.

8. Conclusion

In this paper, two RL based algorithms to solve Economic Dispatch problem was presented. The algorithms were validated using different test cases found in literature, and the performance were found to be promising. In the proposed solution strategy, with a single learning task the schedule for any load demand can be easily retrieved. This makes the algorithm efficient compared to other soft computing techniques. Also in the reinforcement Learning solution, the reward function which in this case is the cost of generation need not be a mathematically defined function. It can take stochastic cost data from a real time environment. In other words the algorithm is suitable to accommodate any cost characteristics.

RL approach to Economic Dispatch is a promising area of research work. With further work, RL based algorithms can be used to learn optimum schedule from real time data. Reinforcement Learning approach is a powerful tool for solving optimisation problem. Developments in RL could be exploited to develop better algorithms for various optimization problems in power systems.

Acknowledgment

The second author gratefully acknowledges the support of All India Council of Technical Education under the TAPTEC scheme (Ref. No. 8021/RID/NPROJ/TAP-139/2002-03).

References