1

Introduction

Multi-agent Coordination by Reinforcement Learning and Evolutionary Algorithms

This chapter provides an introduction to the multi-agent coordination by reinforcement learning (RL) and evolutionary algorithms (EAs). A robot (agent) is an intelligent programmable device capable of performing complex tasks and decision-making like the human beings. Mobility is part and parcel of modern robots. Mobile robots employ sensing-action cycles to sense the world around them with an aim to plan their journey to the desired destination. Coordination is an important issue in modern robotics. In recent times, researchers are taking keen interest to synthesize multi-agent-coordination in complex real-world problems, including transportation of a box/stick, formation control for defense applications, and soccer playing by multiple robots by utilizing the principles of RL, theory of games (GT), dynamic programming (DP), and/or evolutionary optimization (EO) algorithms. This chapter provides a thorough survey of the existing literature of RL with a brief overview of EO to examine the role of the algorithms in the context of multi-agent coordination. The study includes the classification of multi-agent coordination based on different criterion, such as the level of cooperation, knowledge sharing, communication, and the like. The chapter also includes multi-robot coordination employing EO, and specially RL for cooperative, competitive, and their composition for application to static and dynamic games. The later part of the chapter deals with an overview of the metrics used to compare the performance of the algorithms in coordination. Two fundamental metrics of performance analysis are defined, where the first one is required to study the learning performance, while the other to measure the performance of the planning algorithm. Conclusions are listed at the end of the chapter with possible explorations for the future real-time applications.

Multi-agent Coordination: A Reinforcement Learning Approach, First Edition.
Arup Kumar Sadhu and Amit Konar.

Published 2021 by John Wiley & Sons, Inc.

1.1 Introduction

A robot is an intelligent and programmable manipulator, targeted at developing the functionality similar to those of a living creature [1]. It can serve complex and/or repetitive tasks efficiently. Based on the ability of locomotion, robots are categorized into two basic types: fixed base robots and mobile robots. Depending upon the type of locomotion, mobile robots are categorized into three types: wheeled/legged robots, winged/flying robots, and underwater robots, where for the last one, locomotion is controlled by water thrust. In this chapter, we would deal with wheeled robots only.

Agency is a commonly used jargon in modern robotics [1]. An agent is a piece of program/hardware that helps a robot to serve a directed goal. Like humans, when complexity of the problem grows, collective intelligence of the agents is required to achieve the target. The book is on collective/group behavior of agents, who can sense and act rationally. On occasions, agents can share the sensory information or its decision with its teammates directly through a communication network or by displaying its gestural/postural patterns, carrying a specific signature, to communicate a message to its team members.

Communication is a vital issue to generate plans by the agents. However, communication is time-costly and thus is often disregarded for real-world robotic applications. In the present book, we attempted to learn the agent behavioral patterns by a process of learning, and thus avoid communication overhead in real-time planning [1].

There exists quite a vast literature on planning algorithms [2–29]. One of the early robot planning algorithms is due to Nilsson in connection with his research on reasoning-based planning undertaken in Stanford AI research laboratory, which later was adopted in STRIPs [30–32]. In late 1980s to early 1990s, several planning algorithms, including A^* [31, 32], Voronoi diagrams [33], Quad tree, and potential field [34] were evolved. These algorithms presume static world. At the beginning of the 1990s, Michalewicz in one of his renowned papers introduced genetic operators to undertake dynamic planning with local adaptation in trial solutions by specialized mutation operators. The period 1990–2000 has seen significant changes in the planning algorithm with the introduction of supervised/unsupervised neural learning in planning algorithms [32]. The neural algorithms worked in both static and dynamic environments. Typically, in dynamic environments, they predict the direction and speed of motion to determine possible avoidance of collisions. However, they had limited learned experience, and thus were unable to handle planning in the presence of random motions of dynamic obstacles/persons in the environment. Almost at the beginning of the first quarter of the

1990s, Sutton proposed RL algorithm [35], which can help the robot learn its environment through semi-supervised learning. We would deal with multi-agent RL (MARL) in this chapter.

Planning and coordination are two closely used terms in multi-agent robotics [30]. While planning is concerned with determining the sequence of steps to achieve a goal, coordination refers to skillful interaction among the agents to serve their individual short-run/long-run purposes. Apparently, coordination among the agents is required to implement the steps of planning. In centralized planning, the agents need not require coordination, as the central manager takes care of all the agents' states as if its own state and generate a planning cycle by taking care of all the agents' states and goals jointly. Unfortunately, centralized planning is very slow and single-point failure may occur. Thus, centralized planning is not amenable for real-time applications, when the number of agents is excessive. In distributed planning, each agent generates one step of planning by coordinating with other active agents.

Coordination is broadly divided into two types: cooperation [36] and competition [37]. As the names indicate, cooperation requires agents to work hand-in-hand to purposefully serve the common objective of the team. Competition, on the other hand, leads to the success of one team against the failure of its opponent. For instance, in robot soccer, teammates work harmoniously in a cooperative manner, while each team of agents competes for winning at the cost of defeat of the other team.

Researchers are taking keen interest to model agent cooperation/competition by various models/tools. A few of these that need special mention include RL, GT [38–45], DP [46, 47], EO [48–56], and many others [6, 15–28, 57–59]. In RL, agents learn the most profitable joint action at each joint state through a feedback from the environment, and use them for subsequent planning applications [35]. GT requires for strategic analysis in multi-agent domain. In GT, agents evaluate the equilibrium, representative of the most-profitable joint action for the team in a joint state, and execute the joint action for joint state-transition in a loop until the joint goal is explored [38, 41–43, 60]. In DP [46], a complex problem is divided into finite overlapping subproblems. Each subproblem is solved by a DP algorithm and the solution is stored in a database. In the subsequent iterations, if a subproblem already addressed reappears, then that subproblem is not readdressed, but its solution is exploited from the database. In EO algorithm [48, 61–70], the constraint to satisfy the cooperation is checked on the members of the trial solutions before the solutions are entertained for the next generation. Recently, researchers aimed at developing MARL fusing RL, DP, and GT [71, 72]. In this book, we would explore new algorithms of MARL and novel EO.

1.2 Single Agent Planning

In single agent planning [5], an agent searches for the sequence of actions, for which it reaches its predefined goal state from a given state optimally in terms of predefined performance metric. The section describes the single agent planning terminologies and algorithms. Here, single agent planning algorithm includes search-based and learning-based planning algorithms.

1.2.1 Terminologies Used in Single Agent Planning

Definition 1.1 An *agent* [1] is a mathematical entity that acts on its environment and senses the changes in the environment due to its action. The agent is realized by hardware/software means. A hardwired agent has an actuator (motors/levers) and a sensor to serve the purpose of actuation and sensing, respectively.

A learning agent learns its right action at a given location/grid, called state, from its sensory-action doublets. A planning agent identifies its best action at its current state to obtain maximum reward for its action in the given environment.

In a single agent system, the environment includes a single agent. Naturally, the learning/planning steps/moves of the agent is undisturbed by the environment. Figure 1.1 offers architecture of a single agent system.

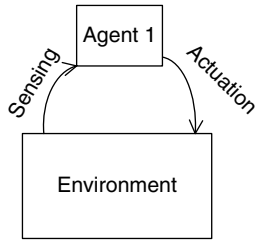


Figure 1.1 Single agent system.

Definition 1.2 The *state* of an agent represents a situation of the agent, concerning the position and/or orientation of the agent in the environment at an instant.

A state-space is a collective set of states of an agent. The definition of the state-space is required *a priori*, to address a planning problem. Such description of the state-space is problem specific. The state-space may be discrete or continuous. We in this book, however, deal with discrete state-space. Figure 1.2 illustrates three discrete states ($s1$, $s2$, and $s3$) of an environment.

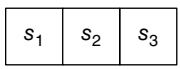


Figure 1.2 Three discrete states in an environment.

Definition 1.3 The *action selection* by an agent is done randomly or using specific strategies, such as ϵ-greedy strategy [35] or the Boltzmann strategy [73]. Random action selection sometimes is inefficient, when the same action is selected repeatedly during the learning phase.

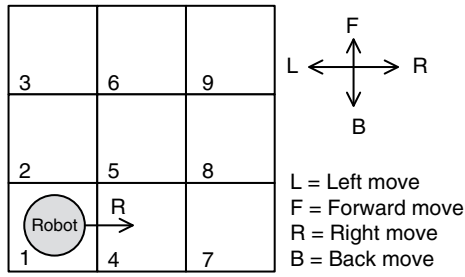


Figure 1.3 Robot executing action Right (R) at state $s1$ and moves to the next state $s2$.

The ε-greedy strategy [35] allows an agent to select random actions from a pool with a probability $=\varepsilon$. For example, if $\varepsilon = 0.2$, then the agent would select 20 actions randomly and 80 greedy actions out of 100 trials from a pool of actions.

Unlike the above, the Boltzmann strategy [73] employs a probability distribution based on the reward function value obtained for individual actions. Usually, an exponential distribution is used to determine the probability of an action in a pool of actions. The larger is the individual reward, the higher is the action selection probability. One control parameter temperature is used to tune the action selection probabilities.

In Figure 1.3, we consider one agent capable of state-transitions using only four actions: Left-move (L), Forward-move (F), Right-move (R), and Back-move (B).

Definition 1.4 A *state-transition* [35] function at state $s \in \{s\}$ due to action $a \in \{a\}$ is a mapping from (s, a) to $s^{/} \in \{s\}$, where $s^{/}$ be a next state, i.e.

$$s^{/} \leftarrow \delta(s, a). \tag{1.1}$$

In deterministic system, for each pair of (s, a), we have a fixed $s^{/}$. In non-deterministic (or stochastic) situation, for each pair of (s, a), we may have different $s^{/}$. Traditionally, non-determinism is handled in an easier way by assigning a probability mass for each state-transition $\delta(s, a)$, such that the sum of the state-transition probabilities is equal to one.

Non-determinism creeps into the system by various ways. For instance, in robot planning application, the condition of floor, such as its "slippery condition" is a guiding factor to determine the transition probabilities.

Suppose, in Figure 1.4, a robot executes an action a at state s and moves to the next state $s^{/}$, receiving an immediate reward $r(s, a)$ as a feedback from the environment. Suppose the floor on which the robot moves on is slippery. In that case, from a state s because of an action a the robot can have more than one

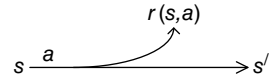


Figure 1.4 Deterministic state-transition.

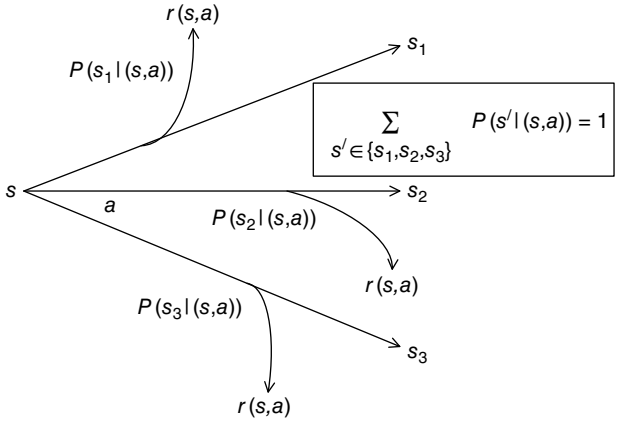


Figure 1.5 Stochastic state-transition.

state-transition, each with a state-transition probability of $P(s^{/} \mid (s, a))$, $s^{/} \in [s_1, s_2, s_3]$, where,

$$\sum_{\forall s^{/}} P\left(s^{/} \mid (s, a)\right) = 1 \tag{1.2}$$

as shown in Figure 1.5. For each state-transition, the agent receives an individual immediate reward $r(s, a)$ with its corresponding state-transition probability.

Definition 1.5 A *policy* [35] π is a decision-making mapping function, representing the probability assignment to a set of actions $\{a\}$ at a given state $s \in \{s\}$ such that, $\sum_{\forall a} \pi(s, a) = 1$, i.e.

$$\pi : s \times \{a\} \rightarrow [0, 1], \tag{1.3}$$

subject to

$$\sum_{\forall a} \pi(s, a) = 1 \tag{1.4}$$

holds for each state s.

In Figure 1.3, at state s_1 there is a set of finite possible actions: L, F, R, and B. Now, random selection of an action from this finite set infinite times results in a policy, $\pi(s_1, a) = 0.25$, $a \in \{L, F, R, B\}$.

In a planning problem, an agent starts by executing its individual action from a predefined state (starting state) with an aim to reach its individual predefined

absorbing state (goal state), optimally in terms of time, path length, energy, and the like. Feasibility and optimality are two desired criterions need to be satisfied while addressing the planning problem [30].

Definition 1.6 *Feasibility* refers to the locomotion of an agent to a feasible next state because of an action form the current state.

Definition 1.7 *Optimality* indicates the performance optimization of the planning algorithm in each step, by minimizing the system resource utilization.

Definition 1.8 The sequence of actions lead to the predefined goal state from a given starting state maintaining the feasibility and optimality jointly in each step is well known as *plan*.

To understand the concept of planning, Example 1.1 is given to realize the movement of a single agent (here robot) in a two-dimensional discrete environment.

Example 1.1 Suppose a robot moves in a two-dimensional 5×5 grid environment as shown in Figure 1.6. There are 25 states and each state is represented by an integer or the Cartesian coordinate (x, y), where $x \in [1, 5]$ and $y \in [1, 5]$. An agent can execute one among the four possible actions $a \in \{L, F, R, B\}$ at a state $s \in [1, 25]$. After executing an action a at a state s, a state-transition takes place and the robot moves from s to the next state $s' \in [1, 25]$ by (1.1). The collection

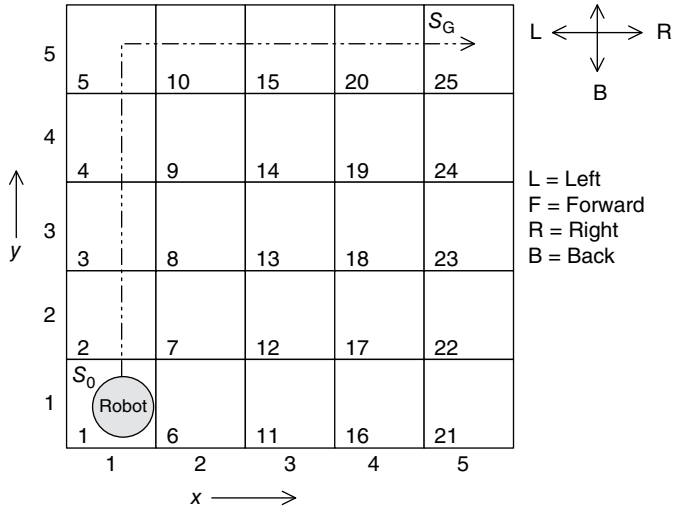


Figure 1.6 Two-dimensional 5 × 5 grid environment.

of state-transitions for which the robot moves from its current state "1" to the goal state "25" is called a feasible path. Among the feasible paths, the optimal one is chosen. One optimal path (here in terms of number of state-transitions) is shown in Figure 1.6 by dotted lines. The example can be made more interesting by adding obstacles in the optimal path.

After finalizing a plan (sequence of actions) by an agent, the agent follows the plan either by execution, refinement, or hierarchical approach.

Execution: In the execution phase, planner's plan is executed in a simulator or by a robot connected to the real environment. There are two types of robots for execution. In the first type, the robot is programmable and acts as an autonomous agent. This approach has the provision of updating the plans after finite time interval. However, most planning algorithms are designed to tackle new situations during the planning phase and hence, the above type of execution is not preferred. The second one is the special-purpose robot designed to solve a specific task given to it.

Refinement: Refinement is the evolution of the planning algorithms toward the better performance as shown in Figure 1.7. In Figure 1.7, agents first compute a collision-free path in the presence of obstacles after that agents optimize (smoothen) the path. Finally, a trajectory is planned following the path and a feedback controller is added for that.

Hierarchical: In hierarchical model, each plan is considered as an action under a larger plan. The same plan may also be defined as a subroutine under the larger plan. In Figure 1.8, the master plan is known as the *root node*. Remaining subsequent plans act as an action for the master plan or plan. There may be infinite number of plans under a master plan or plan. In Figure 1.8, n, m, and p are the real positive integer number. In Figure 1.9 (hierarchical model), agent 1 interacts with environment 1 and agent 2 with environment 2. Again in Figure 1.9,

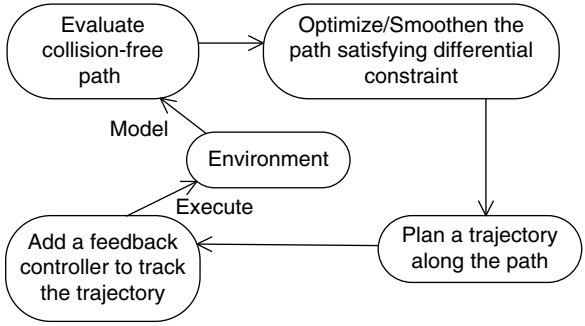


Figure 1.7 Refinement approach in robotics.

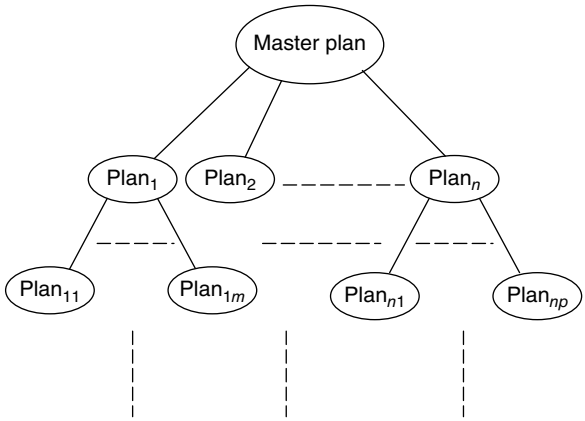


Figure 1.8 Hierarchical tree.

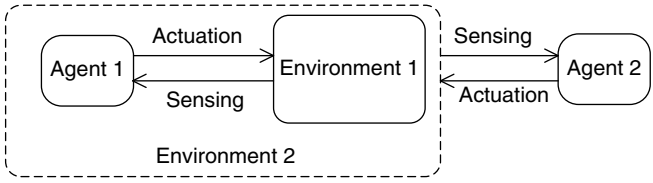


Figure 1.9 Hierarchical model.

environment 2 includes agent 1 and environment 1. So, Agent 2 interacts with the environment 2 as well as 1.

The search-based planning algorithms are employed to evaluate low-cost planning paths in terms of path length, time, energy, and the like, for single-robot planning. The search-based planning algorithms are popular mainly because of their simplicity. The search-based algorithm compromises of the following two parts:

1) In the first part, the realization of the goal following a number of feasible plans is done by employing a search algorithm.
2) The second part is related to the optimal planning, which employs principle of optimality to reduce the computational effort in the planning algorithms.

The search-based planning algorithms avoid the geometric models or differential equations. The search-based algorithms also avoid uncertainty and hence they avoid complications due to probability calculation.

1.2.2 Single Agent Search-Based Planning Algorithms

By search-based planning algorithms, a plan (or sequence of feasible actions) is searched by one of the following methods: forward search, backward search, and bidirectional search [30]. Forward search algorithm deals with the three variant of states. First one is the state which has not been visited yet or the unexplored one is known as *unvisited* state. If all possible state-transitions are explored in a given state, then the state is referred to as a *dead* state. The state which has been visited but still there exist a few unexplored next state is defined as *alive* state. Breadth first [30], Depth first [30], Dijkstra's [74], Best first search [30], Iterative deepening [30], A-star (A^*) [32], and D-star (D^*) [6] are the examples of forward search algorithms. The above forward search algorithms are extendable to the backward search algorithm, by solving the same planning problem by traversing from the goal state to the starting state. The bidirectional search is the combination of forward and backward search. In every search-based planning algorithm, a tree is maintained. For the forward (backward) search, initial (goal) state is the root node of the tree. The advantage of bidirectional search is the radical reduction in the exploration required. In this chapter, only the Dijkstra's, A^* and D^*, and STRIPS like algorithms are discussed as given below.

1.2.2.1 Dijkstra's Algorithm

Dijkstra's algorithm was proposed by computer scientist Edsger W. Dijkstra's [74]. Dijkstra's algorithm is employed to find out the shortest path between two nodes in a graph. In case of robotics, each state is represented by a node of the graph. The starting state is denoted by the source node and instead of finding the shortest path from the source node to all other nodes, the shortest path is obtained from the source node to a specific goal node (goal state of the robot).

The Dijkstra's algorithm is explained for the 3×3 grid shown in Figure 1.10. In Figure 1.10, there are nine states (nodes). State 1 is the source node and state 9 is

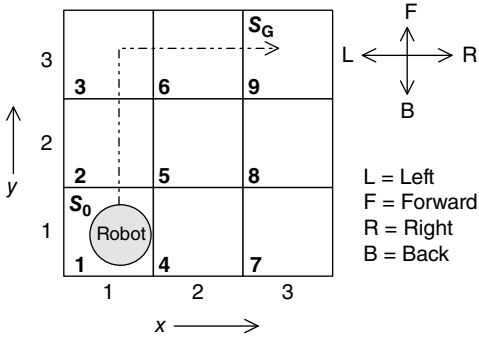


Figure 1.10 Two-dimensional 3 × 3 grid environment.

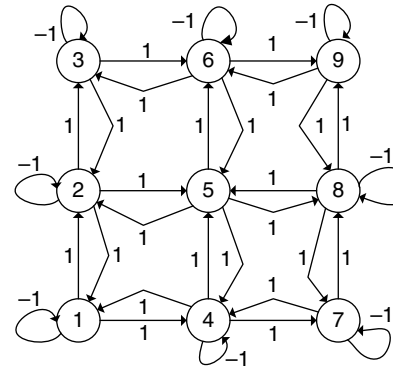


Figure 1.11 Corresponding graph of Figure 1.10.

the goal node. From each node there are maximum four possible paths as shown in the graph (Figure 1.11). Weights of all the edges are 1, ∞, and −1. 1 is assigned for a feasible edge. The self-loop and/or collision between robot and the boundary in Figure 1.11 signify the penalty with reward of −1. ∞ is assigned for an invalid edge. The steps of Dijkstra's algorithm are given in Algorithm 1.1.

The trace of the Dijkstra's algorithm for robot path planning is given in Table 1.1. The bold numbers are the selected node corresponding to the column's node from the current node. The run-time complexity of the Dijkstra's algorithm is $O(|V| \log |V| + |E|)$, where $|V|$ and $|E|$are the number of edges and nodes, respectively.

1.2.2.2 A* (A-star) Algorithm

A* is a heuristic search-based algorithm [32]. In A* algorithm, the quality of a node is measured by introducing two cost functions: one is heuristic cost and another is the generation cost. The heuristic cost, denoted by $h(x)$, is a measure of distance (here city block distance) between the current node x to the goal node. The generation cost of a node x, denoted by $g(x)$, measures the distance of node x from the source node. Total cost function at node x is the summation of $f(x)$ and $g(x)$. The following definitions are required before explaining the A* algorithm [32].

Definition 1.9 A node x is called *open* if the node x has been generated and the heuristic cost $h(x)$ has been computed over it but it has not been expanded yet.

Definition 1.10 A node x is called *closed* if it has been expanded for generating offspring.

The steps of A* algorithm is given Algorithm 1.2. Example 1.2 is given for better understanding of A* algorithm in the perspective of robot path planning problem.

Algorithm 1.1 Dijkstra's Algorithm

Input: Mark all the unvisited nodes and the current node is set as the source node; Generate a search graph G, including the starting node x. Mark node x as an open;
Output: The optimal path;
Begin
Initialize: Set a distance value to all the nodes in the graph. Set zero for the source node (here state 1) and ∞ for the remaining nodes;
Repeat
1) From the current node, explore all the unvisited neighbors and evaluate their distances from the initial node. (For example, let the current node, x has a distance of 3 unit from the source node, and an edge connecting x with another node y has distance of 2. Now, the distance to y through x from the source node becomes 3 + 2 = 5. Compare the currently evaluated distance with the previously recorded distance (∞ at the beginning). If the currently evaluated distance is less than previously recorded distance, then update the database by the currently evaluated distance, otherwise do nothing.
2) Once all the neighbors of the current node have been explored, the current node is marked asvisited (not checked further), and the evaluated distances are recoded as the final and minimal distances.
3) Select one unvisited node with smallest distance as the next current node;
Until goal state reached;
End.

Table 1.1 Trace of Dijkstra's algorithm for Figure 1.11.

		Nodes →								
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Visited ← nodes	{1}	−1	**1**	∞	1	∞	∞	∞	∞	∞
	{1,2}	1	−1	**2**	2	2	∞	∞	∞	∞
	{1,2,3}	2	1	−1	3	2	**3**	∞	∞	∞
	{1,2,3,6}	3	2	1	4	3	−1	5	6	**4**

Algorithm 1.2 A* Algorithm

Input: Generate a search graph G, including the starting node *x*. Mark node *x* as an open;
Output: The optimal path;
Begin
Initialize: Create a list of *closed* node keeping them initially empty;
Repeat
1) If list of *open* node is empty, then exit with failure;
2) Let node *n* is selected from the list of open nodes and removes it from the set. Put the node *n* on the closed nodes list;
3) If *n* is the goal node, then exit and return the solution obtained to trace a path from the node *n* to node *x* in the search graph *G*;
4) Expand node *n* and generate the set *M*, which contains its successors that are not already the ancestors of *n* in *G*. Add the elements of *M* as successors of *n* in *G*;
5) Point *n* from each members of *M*, which does not belong to *G* and add them in the open list. If for all the members of *M* already belong to open or closed list of nodes, then redirect the pointer to *n*, subject to the shortest path is found through *n*. If all the members of *M* are belong to closed list of nodes, then redirect the pointers of its entire offspring in *G*, so that they point toward the back along the best paths found till now to these offspring;
6) Sort the elements of open list in order of increasing cost function (sum of heuristic cost and generation cost);
Until goal state reached;
End.

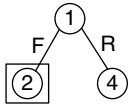
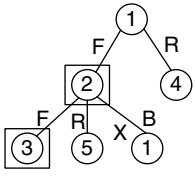
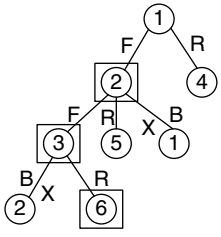
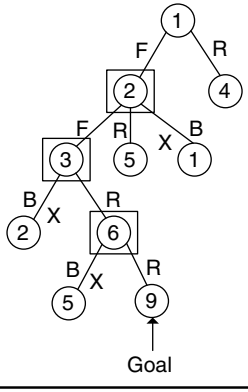
Example 1.2 In this example, the A* algorithm is employed to find the shortest path between source node 1 to the goal node 9 as shown in Figure 1.10. The heuristic cost $h(x)$ of node x (x_x, x_y) is given by the city-block distance and it is defined in (1.5).

$$h(x) = |x_g - x_x| + |y_g - y_x|, \tag{1.5}$$

where (x_g, y_g) is the goal coordinate.

The trace of the A* algorithm is given in Table 1.2. In step 0, robot starts from node 1 and its heuristic cost is 4 and generation cost is 0. Hence, total cost is 4.

Table 1.2 Trace of A* algorithm from Figure 1.10.

Step	State-space		Heuristic cost	Generation cost	Total cost
0			4	0	4
1		For node 2 (**selected**)	**3**	**1**	**4**
		For node 4	3	1	4
2		For node 3 (**selected**)	**2**	**2**	**4**
		For node 5	2	2	4
3			1	3	4
4			0	4	4

The bold values signifies the selected node and its corresponding cost.

In step 1, node 1 is expanded by the action forward (F) to node 2 and by the action right (R) to node 4. The total cost of both the nodes 2 and 4 is $3 + 1 = 4$. Node 2 is selected and it is expended further by the actions forward (F), right (R), and back (B) to the nodes 3, 5, and 1, respectively. The total cost of the nodes 3 and 5 is $2 + 2 = 4$. Node 1 is not selected following Definition 1.10. Node 3 is selected and it is extended further to node 2 and 6 by the action back (B) and right (R), respectively. Here, node 2 is a closed node by Definition 1.10 and hence it is eradicated. So, node 6 is expanded to node 6 and 9 by the action back (B) and forward (F). Again node 6 is eradicated by Definition 1.10 and node 9 is the goal state. The total cost function of node 9 is 4 with 0 heuristic cost. Hence, optimal path is generated by sequentially following the nodes 1, 2, 3, 6, and 9.

1.2.2.3 D* (D-star) Algorithm

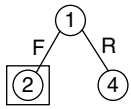
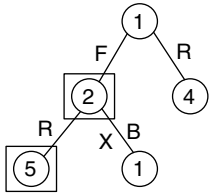
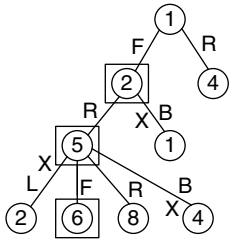
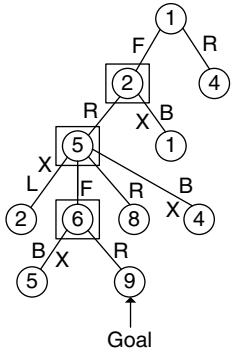
Unlike A* [68], D* [6] algorithm may be employed to efficiently plan in dynamic unknown or partially known environments, by adjusting the weights of the edges (arcs). In the present path-planning application, each state is assumed as a node and weight of each edge (arc) connecting two nodes represents the cost of moving from one node to another. Initially, a path is planned from current node to the goal employing the A* algorithm using the known information. In the journey of the robot toward the goal state, it discovers the presence of obstacles in its path and the graph is modified by adapting the arc weight. The robot again computes the shortest path from its node position to the goal. The process continues until it reaches its goal position or it concludes that the goal is inaccessible. The trace of the D* algorithm is shown below in Table 1.3 by adding an obstacle in state 3 of Figure 1.10 as shown in Figure 1.12. Steps 0 and 1 are same as A* algorithm. In step 2, node 3 is expanded to node 1 and 5 by action right (R) and back (B), respectively.

Node 3 is not accessible as there is an obstacle at node 3. So, node 5 is expanded by left (L), forward (F), right (R), and back (B) actions to nodes 2, 6, 8, and 4, respectively. Nodes 2 and 4 are closed nodes following Definition 1.10. Selecting node 6 and expanding it to nodes 5 and 9 by actions back (B) and right (R), respectively, the goal node 9 is reached. So, optimal path is generated by sequentially following the nodes 1, 2, 5, 6, and 9.

1.2.2.4 Planning by STRIPS-Like Language

Representation is the main bottleneck of the earlier explained search-based planning techniques due to enormous state-space. To address such representation problem, STRIPS-like language [30] is proposed by the Stanford Research Institute Problem Solver group, which is expressive enough to characterize a planning problem logically. STRIPS stands for Stanford Research Institute Problem Solver.

Table 1.3 Trace of D∗ algorithm from Figure 1.12.

Step	State-space		Heuristic cost	Generation cost	Total cost
0			4	0	4
1		For node 2 (**selected**)	**3**	**1**	**4**
		For node 4	3	1	4
2		For node 5	2	2	4
3		For node 6 (**selected**)	**1**	**3**	**4**
		For node 8	1	3	4
4			0	4	4

The bold values signifies the selected node and its corresponding cost.

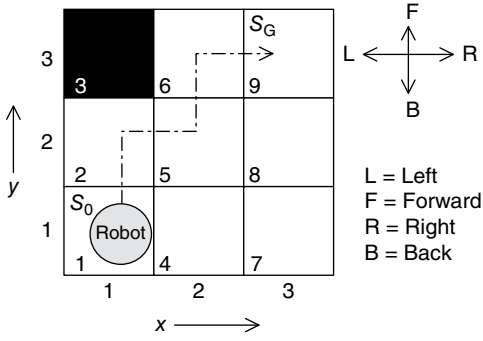


Figure 1.12 Two-dimensional 3 × 3 grid environment with an obstacle.

STRIPS is the first well-known logic-based representation of the discrete planning algorithm, which is the extension of first-order logic (propositional logic). The following representations are employed in STRIPS.

State: In STRIPS, an agent decomposes the environment into logical conditions (TRUE or FALSE), and then the state is represented by conjunctions of function-free ground literals. Ground literal refers to the predicates, which cannot break any more. Suppose, a home service robot is instructed to bring a cup of tea with a biscuit and a magazine. So, in STRIPS, the initial state is formed using the following predicates "at(home)," "┐have(tea)," "┐have(biscuit)," and "┐have (magazine)." Here, home represents the initial position. Now, initial state is the conjunctions of the function-free predicates (ground literals), i.e. "at(home) ^┐have(tea)^┐have(biscuit)^┐have(magazine)." However, the goal state is the "at(home)^have(tea)^have(biscuit)^have(magazine)". Now, the task is to find out the sequence of actions to reach from the initial state to the goal state.

Action: An action follows the following two conditions: preconditions and effect. In precondition, an agent needs to satisfy certain feasibility condition before executing an action. For example, for "have(tea)" the agent must go to a nearby tea stall because tea is not available at(home). Also the preconditions are always positive ground literals. On the other hand, effects are the conjunction of positive and negative ground literals. For example, if there is an action "go(tea stall)" from "at(home)", then the precondition is "at(home)^path(here, there)" and the effect is "at(tea stall)^┐at(home)". Hence, to reach the goal state "at(home)^have(tea)^have (biscuit)^have(magazine)," an agent must satisfy all the preconditions and effects.

1.2.3 Single Agent RL

In the above perspective, learning can assist an agent to select actions. In single agent RL [35, 75–77] (Figure 1.13), an agent receives a reward/penalty as a feedback due to an action at its present (current) state or situation from the surrounding (environment). Such scalar feedback measures the quality of the action in that state. In the literature of RL, this quality value is well known as state–action value. The robot remembers or stores the <state, action, reward> profile as an experience for future reference. Once the robot learns all possible <state, action, reward> profiles, it plans optimally in terms of time and/or energy, from any state within the environment it learns. The single agent RL can be explained by the well-known multiarmed bandit problem [78, 79].

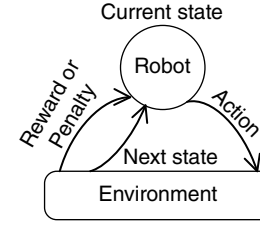


Figure 1.13 Structure of reinforcement learning.

1.2.3.1 Multiarmed Bandit Problem

In the literature of English, a bandit refers to a robber or gambler, who belongs to a gang typically isolated

from the human society. If a bandit has only one arm, then the bandit is called one-armed bandit. The one-armed bandit is also well known as a slot machine, because a slot machine is operated by a button located on the front panel of the machine. A *slot machine* is a casino gambling machine, which rolls three or more times once the button is pushed. As the slot machine stops rolling, it pays off the bandit based on the pattern formed by the symbols visible on the front side of the machine. A multiarmed bandit consists of a series of slot machines arranged in a row. In multiarmed bandit problem [78, 79], the bandit has to decide which slot machine to play and for how many times to maximize the sum of the rewards earned.

The gambler starts playing the multiarmed bandit problem without any knowledge about the slot machines. In each trial, the gambler faces a trade-off between the "exploration" of a new slot machine to obtain better reward than the present rewards, and "exploitation" of a slot machine that has already obtained highest expected rewards. Similar trade-off is experienced by a reinforcement learner in RL. Hence, the multiarmed bandit is employed to manage several projects in a large organization, where initially the properties of the projects are partially known or unknown, but as time passes, the properties becomes fully known to the bandit.

Suppose, a multiarmed bandit [78, 79] has N-slot machines (or N-arms), which are being played by the bandit, and the bandit receives different rewards for each arm, with an aim to determine the arm having the maximum reward. To choose the best arm, i.e. an arm corresponding to the maximum reward (or greedy reward); the agent (or bandit) may compute the running average of rewards of all the arms given in (1.6).

$$Q_t(a) = \frac{r_1 + r_2 + \cdots + r_k}{k}, \tag{1.6}$$

where $Q_t(a)$ refers to the estimated value of the action a in t trials (play). We assume that action a was played k times in t trials and r_k was the reward of choosing the action (arm) a at kth time step. As choosing an arm is analogous to choosing an action, the value of each arm may also be defined as the expected reward of the arm. Let the expected optimal reward of the action a is $Q^*(a)$. Based on the greedy action selection policy, the optimal action a^* is chosen by (1.7).

$$a^* = \underset{a}{\operatorname{argmax}}\, Q^*(a). \tag{1.7}$$

The said greedy action selection may trap the agent (bandit) in local minima. To overcome the problem of trapping in local minima, an agent has to explore a new arm to receive new reward (well known as exploration), which might be better than the present reward. Randomization of the probability of choosing an arm, which is not the greedy one, is referred to as the exploration. In RL, always there is a trade-off between the exploration and exploitation. For example, let $N = 10$ in

the said multiarmed bandit problem. Each arm (analogous to an action) $a \in [1, 10]$ has a random reward given in (1.8) drawn from a normal random distribution with mean zero and variance one, $N(0, 1)$. Equation (1.8) represents the true value or expected reward of the 10 arms.

$$Q^*(a) = [0.0325, 0.8530, 0.1341, 0.0620, -0.2040, 0.6525, 0.8927, -0.9418, -1.4122, 0.8089]. \tag{1.8}$$

By (1.7) and (1.8),

$$\begin{aligned} a^* &= \underset{a}{\operatorname{argmax}} Q^*(a) \\ &= \operatorname{argmax}[0.0325, 0.8530, 0.1341, 0.0620, -0.2040, 0.6525, 0.8927, \\ &\quad -0.9418, -1.4122, 0.8089] \\ &= 7. \end{aligned} \tag{1.9}$$

By (1.9), seventh action is the optimal action denoted by a^*. The learning process is started by estimating the true values from the earlier distribution of $N(0, 1)$ setting the exploration parameter ε to 0.2 and the first estimate is given in (1.10).

$$Q^0_{est}(a) = [0.6761, -1.4321, -0.1824, 3.1140, -1.5285, -2.4264, -1.6687, -0.5252, -0.1021, -0.7124]. \tag{1.10}$$

By (1.7), (1.10), and assuming $Q^*(a) = Q^0_{est}(a)$,

$$\begin{aligned} a^* &= \underset{a}{\operatorname{argmax}} Q^*(a) \\ &= \operatorname{argmax}[0.6761, -1.4321, -0.1824, 3.1140, -1.5285, -2.4264, \\ &\quad -1.6687, -0.5252, -0.1021, -0.7124] \\ &= 4. \end{aligned} \tag{1.11}$$

By (1.11), the bandit should choose the fourth action but by (1.9), the optimal action is the seventh one. So, the greedy choice is misleading the action selection. Several estimations are done to update the $Q_{est}(a)$ vector using (1.6). The learning process continues until the agent recognizes the seventh action as its best choice among the 10 actions. The variation of average reward with the number of trial for different ε is given in Figure 1.14.

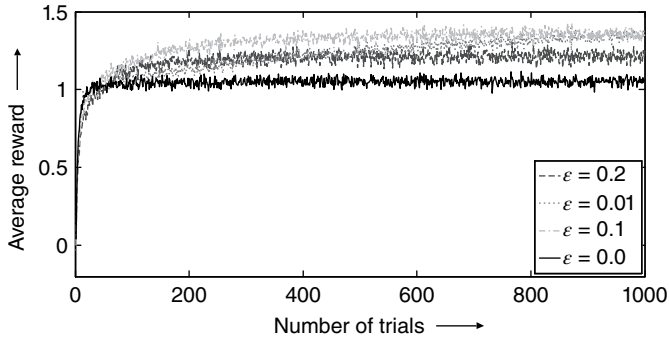


Figure 1.14 Variation of average reward with the number of trial for different ε in 10-armed bandit problem.

1.2.3.2 DP and Bellman Equation

DP [46] is an optimization technique, which transforms a large complex problem into a sequence of simple problems, by dividing it into finite overlapping subproblems, where overlapping indicates that the subproblems can recursively form the actual large complex problem. Breaking a large complex problem into finite overlapping subproblems is the condition of applying DP upon the large complex problem. In DP, there is a relation between the solution offered by the large problem and solutions offered by the subproblems. In the literature of optimization, this relationship is well known as the Bellman equation (BE) or DP equation.

Each subproblem is solved by a DP algorithm and the solution is stored in a database. In the subsequent iterations, if a subproblem already addressed reappears, then that subproblem is not readdressed, but its solution is exploited from the database. Finally, one optimal solution is chosen from the evaluated value functions. The basic four steps for a DP algorithm are given below [46].

1) Divide the large complex problem into finite overlapping subproblems.
2) A value function is defined recursively based on the overlapping subproblems.
3) Compute and memorize the value functions of the overlapping subproblems to avoid repetition.
4) Obtain an optimal solution from the evaluated value functions.

Value function is the heart of DP, as it expresses the quality of a state because of an optimal action in terms of numerical value. If one needs to maximize the value function $v(s)$ at a state $s \in \{s\}$, then using the principle of DP, the problem can be expressed in the BE as given in (1.12).

$$v(s) = \max_a \left[r(s,a) + \gamma v\left(s^{/}\right) \right], \quad \gamma \in (0,1), \tag{1.12}$$

where $r(s, a)$ refers to the reward received at state s because of an action a and $v(s^{/})$ denotes the value function at next state $s^{/}$.

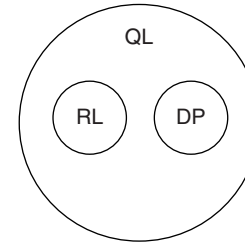


Figure 1.15 Correlation between the RL and DP.

1.2.3.3 Correlation Between RL and DP

It is apparent from the earlier sections that the RL works on the principle of reward/penalty received by the agents as a feedback from the environment, DP is nothing but an optimization technique, which optimizes the BE [71, 72]. Figure 1.15 indicates that the single agent Q-learning (QL) is the combination of the RL and the DP. The details of single agent Q-learning are given in the next section.

1.2.3.4 Single Agent Q-Learning

Q-learning is one well-known paradigm among the RL techniques coined by Watkins and Dayan [80] in 1989. In Q-learning, an agent (robot) adapts in an unknown environment, and receives two types of rewards due to an action at a given state within the environment. One reward is immediate reward received as a feedback from the environment as explained earlier in Section 1.2.3. Another reward is evaluated at the next state. The evaluated reward at the next state is of two types based on the nature of the environment. If the environment is deterministic, then best (or optimal) future reward is evaluated at the next state, shown in Figure 1.16. Since, in deterministic environment, an agent can move from a given state to the next state with probability one due to an action. On the other hand, in stochastic environment, a robot moves from a given state to the next one by assigning a probability in [0, 1] due to an action. Hence, in stochastic environment, the robot evaluates the expected best (or optimal) future reward at the next state. The expected best future reward in the next state is the expectation of selecting the best action in the next state in terms of numerical value. The mechanism of evaluating the expected best future reward in Q-learning is shown in Figure 1.17.

In Figure 1.16, initially all the Q-values at Q-table are set to zero. At the current state 1, the robot executes an action right (R) and receives an immediate reward $r(1, R) = 0$ from the environment. In the next state 3, maximum future reward is evaluated from the Q-table. Until $(3, F)$ is not explored, in the next state, $Q(3, L) = 81$ is the best future reward and updated Q-value at $(1, R)$ is $Q(1, R)$=72.9. On the other hand, once $(3, F)$ is explored in the next state, $Q(3, F) = 100$ is the best future reward and updated Q-value at $(1, R)$ is $Q(1, R) = 90$.

In Figure 1.17, "R" inside the circle symbolizes a robot. Here, each state 1 to 3 has distinct frictional properties. In such stochastic environment, any one next state among the three possible next states may be reached due to left action executed

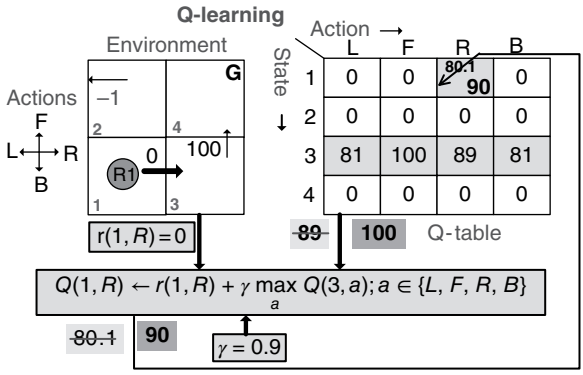


Figure 1.16 Single agent Q-learning.

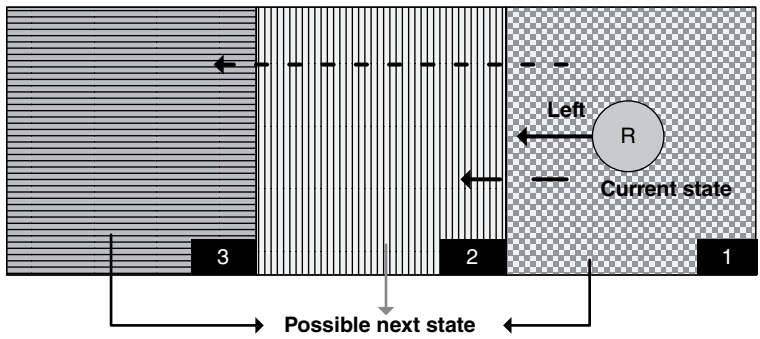


Figure 1.17 Possible next state in stochastic situation.

by R at 1. So, there is a probability of moving to the next state from the current state due to an action. In the literature, such probability is well known as the state-transition probability and the expected future reward is evaluated thereof.

In Q-learning, the future reward prediction depends on the current state–action pair. It is apparent that the future reward prediction in Q-learning of an agent depends exclusively upon the current state but not on the past state–action pairs, which is the Markov property. The Markov property is also well known as the memoryless property. This idea is framed inside the Markov Decision Process (MDP). In Q-learning, the MDP plays a significant role in finding the optimal value function corresponding to the optimal policy π^*. The definition of the MDP is given in Definition 1.11 [81].

Definition 1.11 *A MDP* is a 4-tuple $\langle S, A, r, p\rangle$ [82], [83], where S refers to a finite set of states, A denotes a finite set of actions, $r : S \times A \rightarrow \mathbb{R}$ refers to the reward function of the agent, and $p : S \times A \rightarrow [0,\ 1]$ indicates the state-transition probability.

$$v(s, \pi^*) = \max_a \left[r(s, a) + \gamma \sum_{s'} p\left[s' \mid (s, a)\right] v\left(s', \pi^*\right) \right], \tag{1.13}$$

where $v(s, \pi^*)$ and $v(s', \pi^*)$ represent the value at current state s and next state S' due to optimal policy π^*, γ denotes the discounting factor, $p[s' \mid (s, a)]$ is the state-transition probability to reach next state s' from current state s due to action $a \in A$, and $r(s, a)$ is the immediate reward at state s due to action a.

If an agent directly learns its optimal strategy without knowing either reward function or the state-transition probability, then the learning policy is called model-free RL [84]. Q-learning is one such model-free learning, involving the basic equation given in (1.14).

$$Q^*(s, a) = \left[r(s, a) + \gamma \sum_{s'} p\left[s' \mid (s, a)\right] v\left(s', \pi^*\right) \right]. \tag{1.14}$$

Here, $Q^*(s, a)$ is the optimal Q-value. After infinite revisit of state S due to action a, $Q(s, a)$ turns to $Q^*(s, a)$. If next state is deterministically known for each action, then the Q-learning is called deterministic. In deterministic situation, $p[s' \mid (s, a)] = 1, \forall s'$.

Combining (1.13) and (1.14), we can write,

$$v(s, \pi^*) = \max_a [Q^*(s, a)]. \tag{1.15}$$

Hence, the problem transforms to determining $Q^*(s, a)$ for all (s, a). If $Q^*(s, a)$ is found, one can identify the action which maximizes the $v(s, \pi^*)$. So, the Q-learning update rule becomes,

$$Q(s, a) = r(s, a) + \gamma \max_{a'} Q\left[\delta(s, a), a'\right], \tag{1.16}$$

where

$$s' \leftarrow \delta(s, a) \tag{1.17}$$

be the state-transition function. Hence, $Q(s', a') = Q(\delta(s, a), a')$. By combining (1.16) and (1.17), we obtain (1.18).

$$Q(s, a) = r(s, a) + \gamma \max_{a'} Q\left(s', a'\right), \tag{1.18}$$

Algorithm 1.3 Single Agent Q-Learning

Input: Current state s and action set A;
Output: Optimal Q-value $Q^*(s, a), \forall s, \forall a$;
Begin
Initialize: $Q(s, a) \leftarrow 0, \forall s, \forall a$ and $\gamma \in [0, 1)$;
Repeat
Select an action $a \in A$ randomly and execute it;
Receive an immediate reward $r(s, a)$;
Evaluate next state $s' \leftarrow \delta(s, a)$;
Update: $Q(s, a)$ by (1.19) for deterministic situation,
by (1.20) for stochastic situation and $s \leftarrow s'$;
Until $Q(s, a)$ converges;
Obtain: $Q^*(s, a) \leftarrow Q(s, a), \forall s, \forall a$;
End.

where $\max_{a'} Q(s', a')$ indicates the action $a' \in A$ for which maximum Q-value, $Q(s', a')$ is received at next state s'. Now, the Q-learning update rule with learning rate $\alpha \in (0, 1]$ is given by (1.19).

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha\left[r(s,a) + \gamma Q\left(s', a^*\right)\right]. \tag{1.19}$$

However, in the stochastic situation, the state-transition probability to reach the next state $s' \in \{s\}$ from the state s because of action a, is $P[s' \mid (s, a)] \neq 1$. So, Q-value adaption rule in the stochastic situation is given by (1.20) [84]:

$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha\left[r(s,a) + \gamma \sum_{s'} P\left[s' \mid (s,a)\right] \max_{a'} Q\left(s', a'\right)\right]. \tag{1.20}$$

After infinite revisit of (s, a), Q-value, $Q(s, a)$ turns to the optimal Q-value $Q^*(s, a)$. The convergence proof of (1.20) is given in [2]. Single agent Q-learning steps are given in Algorithm 1.3.

1.2.3.5 Single Agent Planning Using Q-Learning

Figure 1.18 explains the single robot planning mechanism. At first, the robot (R1) observes its current state 3 and its corresponding Q-values from the Q-table. Then at 3, the robot evaluates the action corresponding to the maximum Q-value using

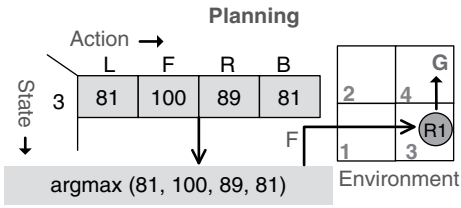


Figure 1.18 Single agent planning.

the learned Q-table. In Q-table, at the row of 3, the action R corresponds to the maximum Q-value. Robot executes the action R and moves to the next state 4. The above steps are repeated until the robot reaches it goal state.

1.3 Multi-agent Planning and Coordination

Multi-agent planning and coordination are two almost similar terminologies both belonging to the multi-agent systems. Multi-agent planning refers to determining the sequence of feasible actions of the agents to achieve individual goals maintaining optimality. However, coordination refers to skillful and effective interaction among the agents to serve the purpose of all the agents. The section describes the multi-agent planning and coordination terminologies with corresponding algorithms.

1.3.1 Terminologies Related to Multi-agent Coordination

A multi-agent system (MAS) includes several agents. Naturally, the action of an agent influences the rewards received by the other agents. This calls for special arrangement for learning and planning in a MAS. Figure 1.19 outlines the architecture of a MAS.

In the MAS, a state-space is a collective set of states of an agent. The definition of the state-space is required a priori, to address a planning problem. Such description of the state-space is problem specific. In a multi-robot coordination problem, instead of states, the joint state is defined.

Definition 1.12 A *joint state* is the collection or union of individual states in a fixed order following the ascending order of the agents.

Suppose s_i is the individual state of agent $i \in [1, m]$, then the joint state for m agents system is given by $S = < s_i >_{i=1}^{m}$.

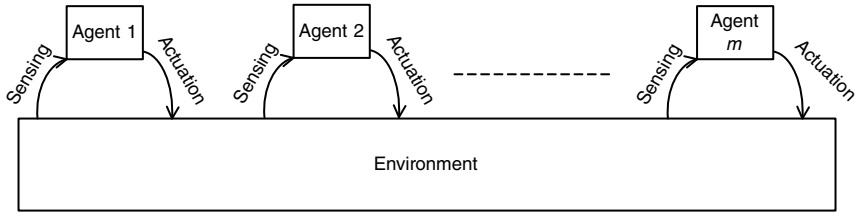


Figure 1.19 Multi-agent system with m agents.

Definition 1.13 The phrase: *joint-action* is a widely used term in the MAS and is defined by the collection or union of individual actions in a fixed order following the ascending order of the agents.

Suppose a_i is the individual action of agent $i \in [1, m]$, then the joint action for the m agent system is given by $A = < a_i > _{i=1}^{m}$. In Figure 1.20, due to joint action <R, L> at <1, 8> robots move to the joint next state <4, 5> as shown in Figure 1.20.

1.3.2 Classification of MAS

There exist several state-of-the-art attributes, based on which the MAS is classified [1, 37]. Attributes relevant to the present book are employed here to classify the MAS. Basically MAS is of two types: cooperative and competitive. Like any social living beings, an agent belonging to cooperative MAS cooperate with remaining agents. However, in the competitive MAS, agents do compete among themselves to acquire limited resources required for livelihood. In this chapter, we consider only the cooperative MAS.

Classification based on cooperation: Classification of MAS based on the cooperative aspect of the agents is done by measuring the ability of an agent to cooperate with remaining agents while performing a task. Agents cooperate with the

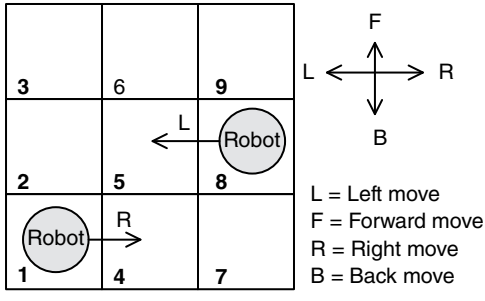


Figure 1.20 Robots executing joint action <R, L> at joint state <1, 8> and move to the next state <4, 5>.

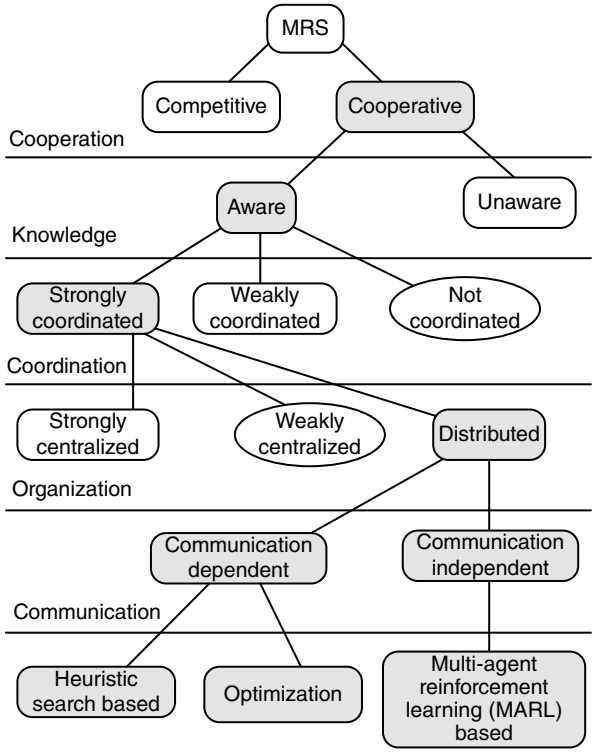


Figure 1.21 Classification of multi-robot systems.

remaining agents are well known as cooperative agent and those do not cooperate rather they compete with others are distinguished as noncooperative agent. The goal of cooperative agents is to achieve a common objective. On the other hand, the noncooperative agents have always conflicting objectives. Figure 1.21 provides a detailed classification of cooperative agents only based on the knowledge level.

Classification based on knowledge level: Further classification of cooperative MAS can be done based on the knowledge level of an agent about the remaining agents in the same team. In Figure 1.21, one is aware agent, which has knowledge about its teammates and the unaware agent does not have such knowledge about the remaining agents in the environment.

Classification based on coordination: Next, the aware agents are classified based on the coordination procedure employed by the agents. There are three types of coordination. In strong (weak) coordination, agents strictly (do not strictly) follow the coordination protocols. In the third type, i.e. not coordinated, agents do not

coordinate with other agents. Classification based upon the coordination is shown in Figure 1.21.

Classification based on organization: Strongly coordinated agents are further classified based on the responsibilities of the agents in a team (or organization) as shown in Figure 1.21. By this aspect, the centralized approaches are distinguished from the distributed approaches. In the centralized approach, an agent is elected as a leader for the entire team. The leader is responsible to distribute the task among all the agents in the team. The remaining agents (follower) act according to the instructions provided by the leader. However, in the distributed system, agents are completely autonomous in view of the decision-making process, as there is no leader in the team. On the other hand, the centralized system can further be classified based upon the way the leader is elected among the team members. If only one robot leads the complete mission, then such centralized system is known as strongly centralized. However, in a weakly centralized system, more than one agent is allowed to lead the team toward the completion of the mission.

Classification based on communication: Distributed robots are classified based upon the dependency on communication among the agents as shown in Figure 1.21. There are two types of distributed agents, one is communication dependent, and another is communication independent.

Besides the above classification, the MAS can further be classified considering "Team Composition" (combination of heterogeneous and homogeneous robots), "System Architecture," and "Team Size."

Several approaches are available in the literature of multi-robot coordination. Among them, coordination by MAQL and EO algorithms are described in this chapter. To improve readability, GT and DP are briefly described below.

1.3.3 Game Theory for Multi-agent Coordination

GT formally analyzes the strategic situation of the MAS, where each agent potentially affects the interests of other agents in the environment [38, 41, 42]. Two types of game are considered in the present book: static and dynamic. The definitions of static and dynamic games are given below.

Definition 1.14 A *static game* with m player is defined by a tuple $\langle m, A_1, A_2, ..., A_m, r_1, r_2, ..., r_m \rangle$ [42], where $A_i, i \in [1, m]$ is the set of finite actions of player i and $r_i : \times_{i=1}^{m} A_i \rightarrow \mathbb{R}, i \in [1, m]$ refers to the reward function of player i, where, $\times$ denotes the Cartesian product.

Definition 1.15 If a static game is played repeatedly, then the game is well known as *repeated game*.

In static game, multiple agents execute their actions at a joint state and agents do not have any state-transition. Hence, a static game is also known as state-less game. Now, to handle the games with state-transitions, another version of game called dynamic game is defined below.

Definition 1.16 A *dynamic game* with m number of agents is defined as a 5-tuple $\langle m, \{S\}, \{A\}, r_i, p_i \rangle$ where $\{S\} = \times_{i=1}^{m} S_i$ is the joint state-space, $\{A\} = \times_{i=1}^{m} A_i$ is the joint action-space, $r_i = \{S\} \times \{A\} \rightarrow \mathbb{R}$ is the reward function at joint state–action of agent i, and $P_i = \{S\} \times \{A\} \rightarrow [0, 1]$ is state-transition function of agent i.

Suppose an agent $i \in [1, m]$ selects an action a_i from the pool of its action set A_i and plays the repeated game. The conjunction of the individually chosen actions for all the agents form a joint action $A \in \times_{i=1}^{m} A_i$. Let $\pi_i(a_i)$ refers to the probability of selecting an action $a_i \in A_i$ by agent i, where

$$\pi_i : A_i \rightarrow [0, 1]. \tag{1.21}$$

If $\pi_i(a_i) = 1$, then the strategy of agent i, π_i is deterministic for $a_i \in A_i$. The strategy profile for m agents is given by

$$\Pi = \{\pi_i : i \in [1, m]\}. \tag{1.22}$$

The strategy profile

$$\Pi_{-i} = \{\pi_1, ..., \pi_{i-1}, \pi_{i+1}, ..., \pi_m\} \tag{1.23}$$

denotes the strategy of all the agents except the strategy of agent i, π_i, where

$$\Pi = \Pi_{-i} \cup \{\pi_i\}. \tag{1.24}$$

It is apparent from Definitions 1.14 and 1.16 that a dynamic game is also a static game with state-transitions. In a static game, agents look for a balanced condition or equilibrium among them, such that no one would receive any incentive by unilateral deviation. In the literature, two well-known equilibria exist: Nash equilibrium (NE) and correlated equilibrium (CE).

Before understanding equilibrium, let at a given state s an agent (here robot) have an action set A. An action $a^* \in A$ corresponding to the maximum reward at state s refers to the optimal or greedy action. Collection of such optimal actions executed at each state is termed as optimal policy or strategy. In a particular state, if a robot executes an action, then the action is well known as a *pure strategy*. However, the *mixed strategy* is the randomization over the pure strategies. To understand mixed strategy, rock-paper-scissor game is given in Example 1.3.

Example 1.3 Rock-paper-scissor [41, 42] is a two player hand game, played for fun by kids and sometimes for decision-making by adults. Each player has three options: rock, paper, or scissor and a player can choose one in a trial. The player

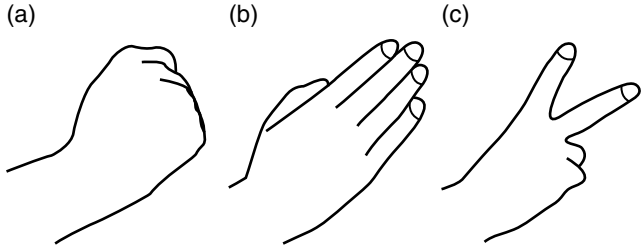


Figure 1.22 Hands gestures in rock-paper-scissor game: (a) rock, (b) paper, and (c) scissor.

expresses his/her choice to another player by using a hand to form one of the shapers as shown in Figure 1.22. With these options, this game can have three possible outcomes excluding a tie. The three possible outcomes are given one by one.

1) One rock crushes scissor. Here, player playing rock beats the player playing scissor.
2) But if paper covers rock, then the player chooses to play paper beats the player playing rock.
3) On the other hand, if scissors cut paper, then the play of paper is defeated by the play of scissor.

However, if the choices of both the players are same, then a tie occurs and the game is replayed until the tie is broken.

After finite trials of the game, the rewards of both the players are given in the reward matrix as shown in Figure 1.23. In Figure 1.23, one cell contains two rewards. The first reward is for Player 1 and second one is for Player 2. In case of a tie, both the players receive 0 reward. If a player wins the game, then the player is rewarded by 1. On the other hand, if the player loses, then the player is penalized by −1.

It is apparent that in the rock-paper-scissor game (Figures 1.22 and 1.23), optimal mixed strategy of Players 1 and 2 is to execute each action with a probability 1/3. Now, suppose Player 1 knows in advance that the Player 2 is playing the pure strategy "paper," then optimal pure strategy for Player 1 is "scissor" as it provides maximum reward to Player 1.

		Player 2		
		Rock	Paper	Scissor
Player 1	Rock	(0,0)	(–1,1)	(1,–1)
	Paper	(1,–1)	(0,0)	(–1,1)
	Scissor	(–1,1)	**(1,–1)**	(0,0)

Figure 1.23 Rock-paper-scissor game.

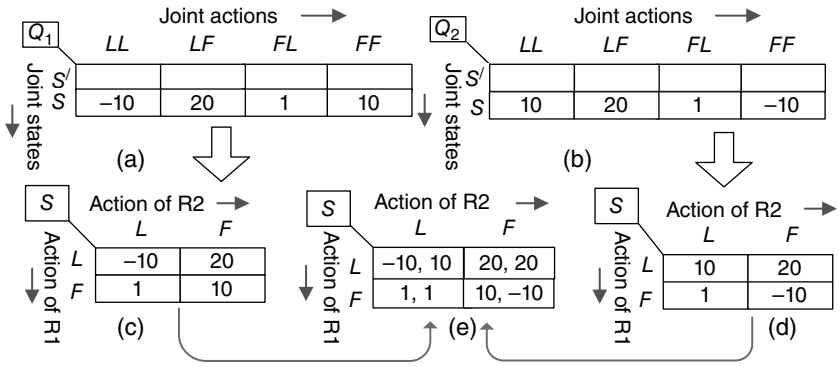


Figure 1.24 Reward mapping from joint Q-table to reward matrix.

1.3.3.1 Nash Equilibrium

NE is a solution concept of the multi-agent interactive system from where no player deviates to maintain its current reward, which is the maximum one. The Definition of NE is given in Definition 1.17. NE is of two types: pure strategy NE (PSNE) and mixed strategy NE (MSNE). To evaluate PSNE at a joint state, an agent selects an action from its own action set, which corresponds to its maximum reward due to joint action, where remaining agents' actions are kept fixed. The joint action at a joint state for which all the agents receive maximum reward and no one has any selfish intension to deviate from its chosen action is well known as PSNE at that joint state. An example is considered in Figures 1.24 and 1.25 to evaluate PSNE in a static game or state-less or one-stage or normal-form game [85].

Definition 1.17 *NE* is a stable joint action (or strategy) at a given joint state (S) of a system that involves m interacting agents, such that no unilateral deviation (deviation of an agent independently) can occur as long as all the agents follow the same optimal joint action $A_N = < a_i^* >_{i=1}^{m}$ at a joint state $S \in \{S\}$ for *PSNE*. Further, for a *MSNE*, agents perform the joint action $A = < a_i >_{i=1}^{m}$ with a probability $p^*(A) = \prod_{i=1}^{m} p_i^*(a_i)$, where $p_i^* : \{a_i\} \rightarrow [0,1], p^* : \{A\} \rightarrow [0,1]$.

Let $a_i^* \in \{a_i\}$ be the optimal action of agent i at s_i and $A^*_{-i} \subseteq A$ be the optimal joint action profile of all agents except agent i at joint state $S = < s_j >_{j=1, j\neq i}^{m}$ and $Q_i(S, A)$ be the joint Q-value of agent i at S because of joint action $A \in \{A\}$. Then the condition of PSNE at S is

$$\begin{aligned} & Q_i\left(S, a_i^*, A^*_{-i}\right) \geq Q_i\left(S, a_i, A^*_{-i}\right), \quad \forall i \\ & \Rightarrow Q_i(S, A_N) \geq Q_i\left(S, A^{/}\right), \quad \forall i \quad \left[\text{where } A_N = < a_i^*, A^*_{-i} > \quad \text{and} \quad A^{/} = < a_i, A^*_{-i} >\right] \end{aligned} \tag{1.25}$$

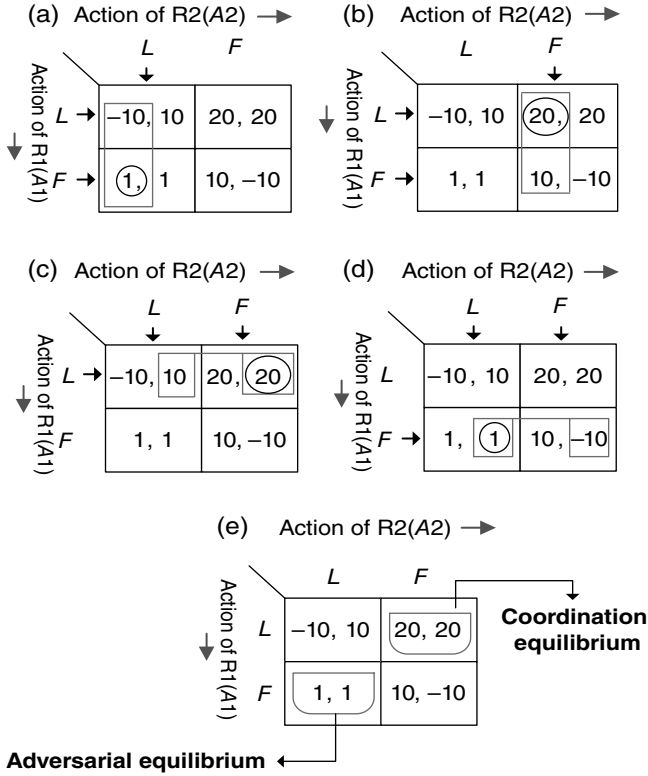


Figure 1.25 Pure strategy Nash equilibrium evaluation. (a) Fix $A1 = L$ and $A2 = L/F$ (b) Fix $A2 = F$ and $A1 = L/F$ (c) Fix $A1 = L$ and $A2 = L/F$. (d) Fix $A1 = F$ and $A2 = L/F$. (e) Nash equilibrium and *FL* and *LF*.

and condition of MSNE at S is

$$Q_i(S, p_i^*, p_{-i}^*) \geq Q_i(S, p_i, p_{-i}^*), \quad \forall i, \tag{1.26}$$

where $Q_i(S,p) = \sum_{\forall A} p(A) Q_i(S,A)$ and $p_{-i}^*(A_{-i}) = \prod_{j=1, j\neq i}^{m} p_j^*(a_j)$ be the joint probability of selecting joint action profile of all agents except agent i denoted by $A_{-i} \subseteq A$.

Agents follow Figure 1.25 to evaluate PSNE $A_N = < a_i^*, A_{-i}^* >$ and Figure 1.26 for MSNE $< p_i^*(a_i), p_{-i}^*(A_{-i}) >$, respectively, at joint state S.

Pure Strategy NE

Assuming in the one-stage game there are two robots R1, R2 and each has two actions. Robot 1 (R1) selects one action from the set {*L*, *F*} and robot 2 (R2) selects one from {*L*, *F*}. As a result there is a joint action set. The joint action set {*LL*, *LF*,

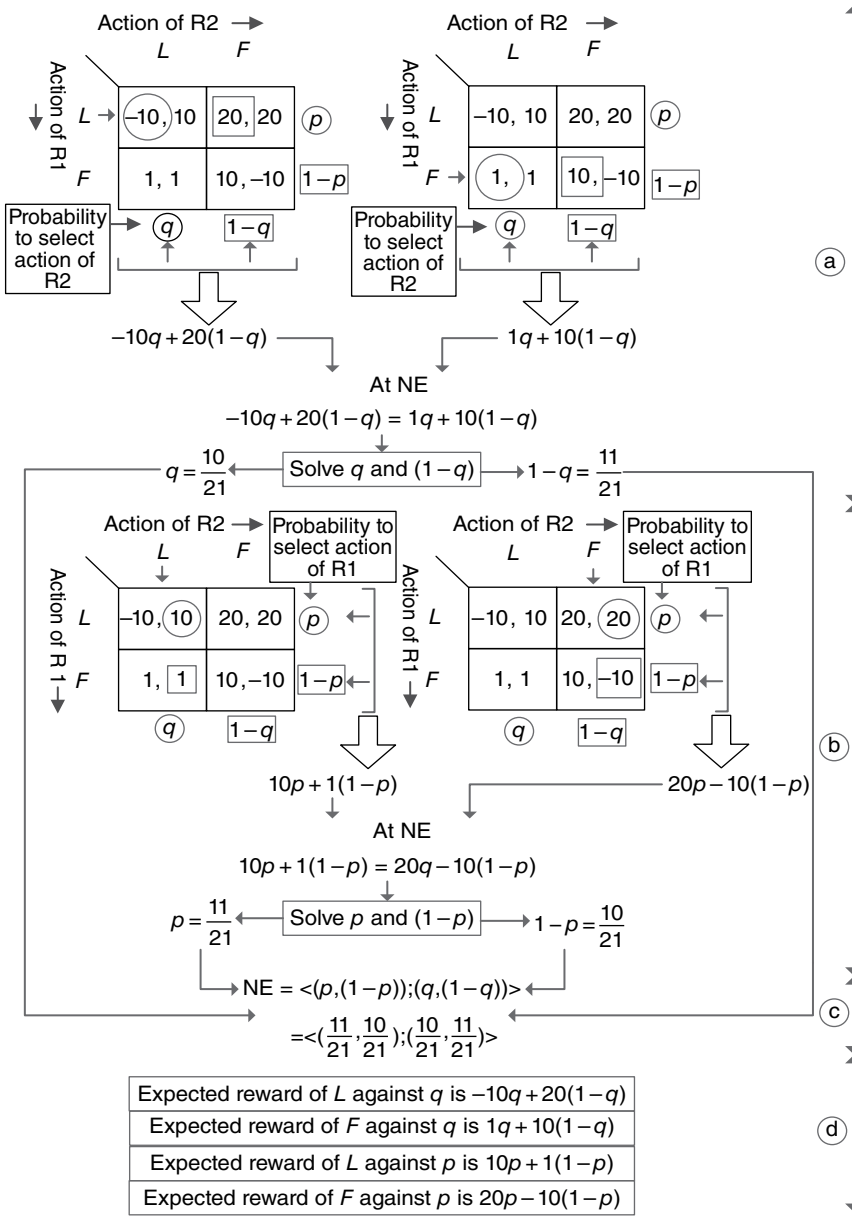


Figure 1.26 Evaluation of mixed strategy Nash equilibrium.

FL, *FF*} is the all possible combinations (the Cartesian product) of {*L*, *F*} and {*L*, *F*}. Figure 1.24a and b provide the reward tables of R1 and R2 at joint state–action space, respectively. The rows of the state–action tables indicate the joint state. Joint state is the conjunction of individual states, here S and S'. Each column corresponds to a joint action A. A can be any one from the set {*LL*, *LF*, *FL*, *FF*}. The entries for each joint state–action pair are called joint state–action value. To evaluate PSNE at joint state S', the rewards at S' due to joint actions are mapped in the reward matrix, as shown in Figure 1.24c and d. In Figure Figure 1.24c and d, each cell of the reward matrices displays the rewards at a joint state S' due to individual actions, respectively, for R1 and R2. In Figure 1.24c–e, rows indicate the actions of R1 and columns indicate the actions of R2. However, in Figure 1.24e, each cell shows the rewards of both the agent. First entry is for R1 and the second one is for R2.

Figure 1.25a–d show the reward matrices of R1 and R2 at joint state S'. In Figure 1.25a, R1 selects its best action assuming that R2 has been selected *L* indicated by solid black arrows. In this situation, R1 prefers action *F* and receives 1 as a reward indicated in Figure 1.25a. Similarly, R1 receives 20 as a reward assuming that R2 has been selected *F* as shown in Figure 1.25b. Similarly, R2 earns 20 and 1, when R1 selects *L* and *F*, respectively, as indicated in Figure 1.25c and d. Finally, Figure 1.25e shows the common solution producing cells, which are the PSNE. Computation of PSNE for two agents is performed by the following three steps:

1) Fix the action of R1, then select the best reward of R2, considering all possible actions of itself.
2) Fix the action of R2, then select the best reward of R1, considering all possible actions of itself.
3) If the results of selection fall in the same cell, then PSNE = joint actions corresponding to the selected common grid.

Here, two PSNE are obtained: <*F*, *L*> and <*L*, *F*>. Let us examine them one by one. For the <*F*, *L*>, both the robots receive 1. Now, if any one robot selfishly attempts to change its action aiming at maximizing its own reward, then the robot, which changes its action, causes to decrease its reward from 1 to −10. Besides, if R1 changes its action from *F* to *L*, then R2's reward improves from 1 to 10. Again, if R2 changes its action from *F* to *L*, then R1's reward improves from 1 to 10. So, the joint action <*F*, *L*> is an adversarial equilibrium. Adversarial equilibrium is a PSNE in competitive situation. Now, the joint action <*L*, *F*> is coordination equilibrium, where both the robots receive maximum reward selflessly, i.e. 20. Coordination equilibrium is a PSNE in cooperative situation. The present book considers coordination equilibrium only.

Table 1.4 Expected reward of R1 and R2 at MSNE.

Expected reward of R1 by employing p against q	$p[-10q + 20(1-q)] + (1-p)[1q + 10(1-q)]$
Expected reward of R2 by employing q against p	$q[10p + 1(1-p)] + (1-q)[20p - 10(1-p)]$

Mixed Strategy NE

MSNE is stochastic. In MSNE, each robot randomizes its own pure strategies by assigning a probability in between zero and one for each pure strategy. Let R1 select its actions L and F with a probability p and $(1-p)$, respectively. Also, let R2 select its actions L and F with a probability q and $(1-q)$, respectively. The summary of rewards at MSNE is given in Figure 1.26. At MSNE, the expected rewards of L and F against q are equal. Equating these two expected rewards yield p and $(1-p)$ as shown in Figure 1.26a. Similarly, one can find q and $(1-q)$ as shown in Figure 1.26b. The expected reward of a mixed strategy is the weighted sum of the expected rewards of all the pure strategies in the mix. Finally, the expected reward of R1 by employing p against q and the expected reward of R2 by employing q against p are given in Table 1.4. Also it is listed in Figure 1.26d. Finally, the MSNE is $<(p, (1-p)); (q, (1-q))>$ given in Figure 1.26c. Table 1.4 provides the expected reward at MSNE for two players. Example 1.4 provides an example of MSNE for a two-player tennis game. Example 1.4 is given below to illuminate MSNE.

Example 1.4 Figure 1.27 shows the reward matrix for a two-player tennis game between Venus and Serena. Let in Figure 1.27, Venus is the row player and Serena is the column player. If Venus chooses Left (L), then she attempts to pass Serena to Serena's left (l). If Venus decides Right (R), then she is attempting to pass Serena to Serena's right (r). Serena chooses l, means that she bends slightly toward her l. Similarly, Serena chooses r means she slightly bends toward her r. There is no PSNE in Figure 1.27. Let's find MSNE for the tennis game. In MSNE, each agent's mix should be the best for the remaining agents' mix. To find Serena's NE mix $(q, 1-q)$, look at Venus's rewards. Now, Venus's rewards against q while choosing

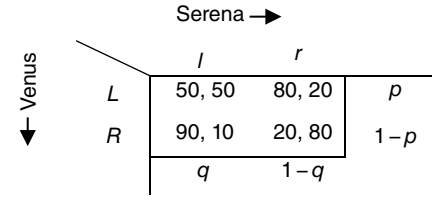


Figure 1.27 Reward matrix for tennis game.

L and R is given by $50q + 80(1-q)$ and $90q + 20(1-q)$, respectively. In MSNE, L and R both themselves must be the best response against q. So,

$$50q + 80(1-q) = 90q + 20(1-q). \tag{1.27}$$

Therefore, solving (1.27), we obtain $q = 0.6$ and $1-q = 0.4$. Now, to find Venus's NE mix $(p, 1-p)$, look at Serena's rewards. Serena's reward against p while choosing l and r is given by $50p + 10(1-p)$ and $10p + 80(1-p)$, respectively. In MSNE, l and r both themselves must be the best response against p. So,

$$50p + 10(1-p) = 10p + 80(1-p) \tag{1.28}$$

Therefore, solving (1.28), we obtain $p = 0.7$ and $1-p = 0.3$. Hence, the MSNE is given by $[(p, 1-p); (q, 1-q)] = [(0.7, 0.3); (0.6, 0.4)]$.

For multi-agent coordination without any communication among the agents, agents face coordination problem in the presence of multiple coordination equilibria [72]. Here, coordination problem refers to the problem of selecting unique equilibrium by all the robots. Such problem can be resolved by selecting a joint action based on a signal (e.g. traffic signal), which is commonly accessible by all the robots. Before discussing about the remedies of equilibrium selection, Example 1.5 is provided to realize the problem.

Example 1.5 The reward matrix for a common reward two-agent static game is given in Figure 1.28, where both a and $b$are the rewards. There are two action sets $\{x0, x1\}$ and $\{y0, y1\}$ for agent X and Y, respectively. Now, if $a > b > 0$, then there are two equilibria $<x0, y0>$ and $<x1, y1>$. But, only $<x0, y0>$ is the optimal and hence, one would expect that the agents play $<x0, y0>$. If $a = b > 0$, then none of the agents have any reason to prefer any one action.

In such situation, there exist multiple equilibria. Choosing one equilibrium among multiple equilibria by random selection or by focusing personal basing may lead to suboptimal (or uncoordinated) equilibrium.

A robot can resolve the problem of equilibrium selection [38] in a coordinated game by repeatedly playing a game by the same robot. In the literature, CE addresses the problem of equilibrium selection.

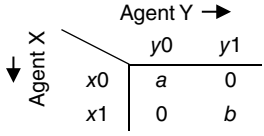


Figure 1.28 Reward matrix of in a common reward two-agent static game.

1.3.3.2 Correlated Equilibrium

CE is more general than the NE [72]. There are four variants of CE: Utilitarian, Egalitarian, Republican, and Libertarian equilibria [72]. In each variant, a numerical value is maximized and its corresponding index (joint action) is well known as CE. The former numerical value may be evaluated by one of the following techniques:

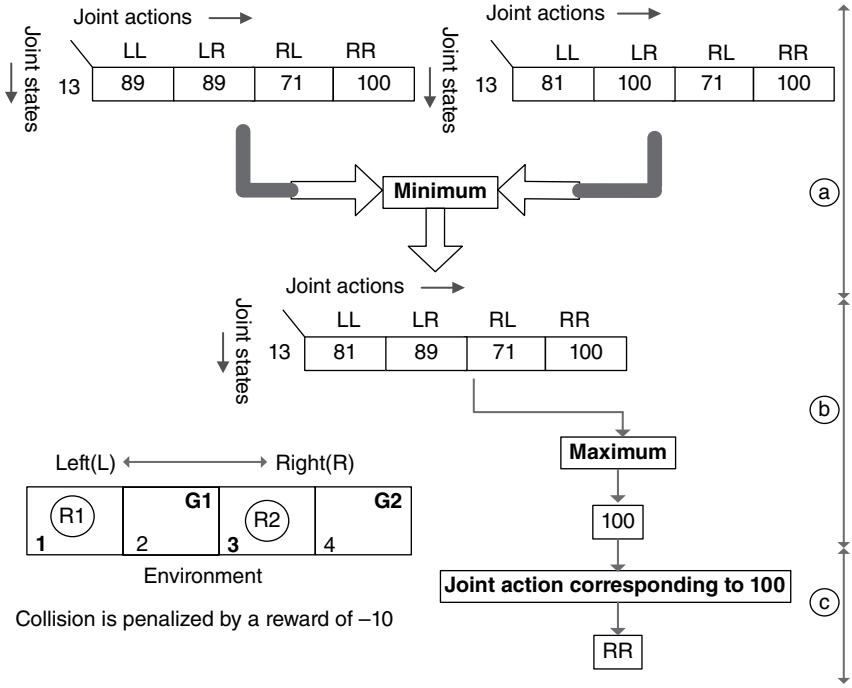


Figure 1.29 Pure strategy Egalitarian equilibrium, which is one variant of CE.

1) In Utilitarian equilibrium, the numerical value to be maximized is evaluated by adding all robots' rewards.
2) The least efficient robot's reward is maximized in the Egalitarian equilibrium.
3) Most efficient robot's reward is maximized in Republican equilibrium.
4) In Libertarian equilibrium, the numerical value to be maximized is evaluated by multiplying all the robots' rewards.

Like NE in CE, there are pure strategy and mixed strategy CE. The definition of CE is given in Definition 1.18. The pure strategy Egalitarian equilibrium evaluation is shown in Figure 1.29.

Definition 1.18 *CE* at a joint state, $S = < s_i >_{i=1}^{m}$ with m interacting agents is the pure strategy CE, A_C and mixed strategy CE, $p^*(A_C)$ if agents follow (1.29) and (1.30), respectively.

$$A_C = \underset{A}{\text{argmax}}[\Phi(Q_i(S,A))], \tag{1.29}$$

$$p^*(A_C) = \underset{p(A)}{\text{argmax}}\left[\Phi\left[\sum_A p(A)(Q_i(S,A))\right]\right], \tag{1.30}$$

where

$$\Phi \in \left\{ \sum_{i=1}^{m}, \mathrm{Min}_{i=1}^{m}, \mathrm{Max}_{i=1}^{m}, \prod_{i=1}^{m} \right\}. \tag{1.31}$$

Game of chicken reflects the idea of CE and is illustrated in Example 1.6.

Example 1.6 In the game of chicken, two players play by heading toward each other as shown in Figure 1.30. If both the players move (M) on the same way, then they collide, which results in penalty for both the agents. If one player moves and another player waits (cooperate (C)), then both the players are rewarded. The player which successfully moves receives more reward, than the player which cooperates. Both the players receive zero reward if none of them move. In Figure 1.31, both the joint action (M, C) and (C, M) is the PSNE. To achieve PSNE without establishing any communication among the players, they should follow a signal (like traffic signal), which is commonly accessible for both the players.

1.3.3.3 Static Game Examples

A few examples of static games are given below.

Constant-sum-game: In constant-sum game [41], summation of the two players' rewards is constant as shown in Figure 1.32, where $<a, b>$ and $<x, y>$ are the action sets of player 1 and 2, respectively. In Figure 1.32, the value of the constant is 1.

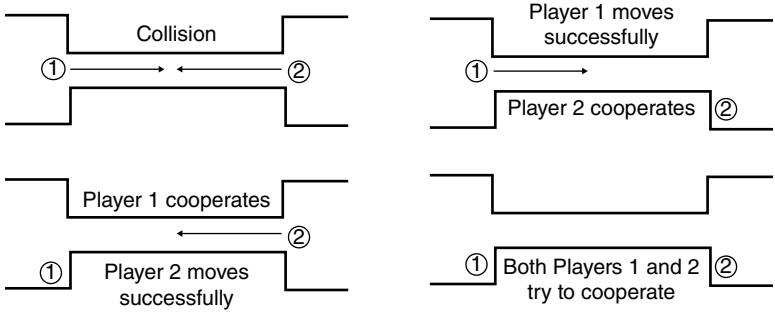


Figure 1.30 Game of chicken.

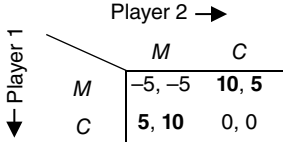


Figure 1.31 Reward matrix in the game of chicken.

Zero-sum game: Zero-sum game is a special case of constant-sum game [41]. It is a two-player game, where the summation of the two players' reward is always zero. This indicates that one player's gain is equivalent to another player's loss. Hence, net change in reward is zero. Chess and tennis are the examples of zero-sum game, where there is a winner and a loser. Financial market is also an example of zero-sum game. In the literature of GT, matching pennies and rock-paper-scissor (given in Example 1.3) are the well-known examples of zero-sum game. Example 1.7 illustrates the game of matching pennies.

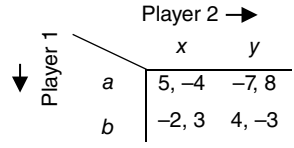


Figure 1.32 Constant-sum game.

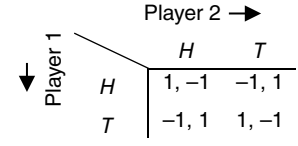


Figure 1.33 Matching pennies.

Example 1.7 In matching pennies, two pennies are thrown by two players simultaneously. The rewards of the players depend on whether the pennies match or not. If both pennies result in head (H) or tail (T), then player 1 wins and rewarded by player 2's penny. If there is a mismatch, then player 2 wins and rewarded by player 1's penny. As one player's gain is other player's loss, hence, matching pennies is a zero-sum game as shown in Figure 1.33. In matching pennies, there is no PSNE instead there exists MSNE.

In some situation, a game does not have a PSNE but every game have a MSNE [42]. For example in the rock-paper-scissor game, there is no PSNE but there exists MSNE.

General-sum-stochastic game: In general-sum-stochastic game, the summation of all the players' rewards is neither zero nor constant. Prisoner's Dilemma is an example of general-sum-stochastic game and is illustrated in Example 1.8.

Example 1.8 In Prisoner's Dilemma, two criminals are suspected of committing a crime and are being interrogated in two separate cells. From human physiology, both the criminals want to minimize their jail sentence. Both of them face the same scenarios as follows (Figure 1.34):

- If Players 1 and 2 each Deny (D) each other, then each of them are sentenced by nine year jails.
- If Player 1 Deny (D) but Player 2 remains Confess (C), then Player 1 will be set free and Player 2 will serve 10 year jails (and vice versa).

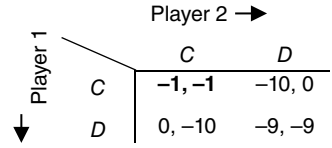


Figure 1.34 Reward matrix in Prisoner's Dilemma game.

- If both Players 1 and 2 remain Confess (C), then both of them will only serve one year jail.

Hence, in Prisoner's Dilemma game, (C, C) is the PSNE.

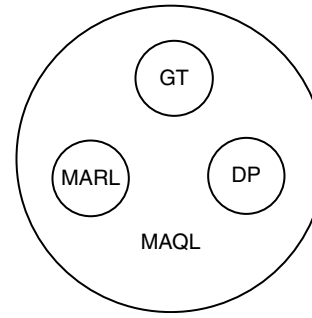


Figure 1.35 Correlation among the MARL, DP, and GT.

1.3.4 Correlation Among RL, DP, and GT

It is apparent from the earlier sections that the RL works on the principle of reward/penalty received by the agents as a feedback from the environment. DP is nothing but an optimization technique, which optimizes the BE. On the other hand, GT helps in analyzing the strategic situation of the agents in the MAS, where an agent significantly affects the interests of other agents in the environment. Figure 1.35 indicates that the multi-agent Q-learning (MAQL) comprising of the MARL, GT, and DP. However, in the literature, MARL is well known as MAQL for simplicity.

1.3.5 Classification of MARL

Based on the task type, MARL is classified as cooperative, competitive, and mixed as shown in Figure 1.36 [85]. Now, the cooperative and mixed algorithms may be designed for static (stateless) games or for stagewise (dynamic) games. However, there are only two competitive algorithms for two agents, namely Minimax-Q and heuristically accelerated multi-agent RL (HAMRL).

Joint Action Learners (JAL) and Frequency Maximum Q-value (FMQ) heuristic are classified as cooperative static algorithm. Team-Q, Distributed-Q, Optimal Adaptive Learning (OAL), Sparse Cooperative Q-learning (SCQL), Sequential Q-learning (SQL), and Frequency of the maximum reward Q-learning (FMRQ) fall within the scope of cooperative dynamic algorithms.

The mixed static algorithms are classified based on the belief on the other agents' policy of an agent and the steps required in searching the optimal policy (direct policy search). Belief-based learning include Fictitious play (FP), Meta Strategy, *AWESOME*, and Hyper-Q. The direct policy search-based algorithms are classified based on variation of the learning rate: fixed learning rate and variable learning rate. Fixed learning rate includes Infinitesimal Gradient Ascent (IGA) and Generalized IGA (GIGA). Win or Learn Fast-IGA (WoLF-IGA) and GIGA-Win or Learn Fast (GIGA-WoLF) are under the variable learning rate. The dynamic mixed

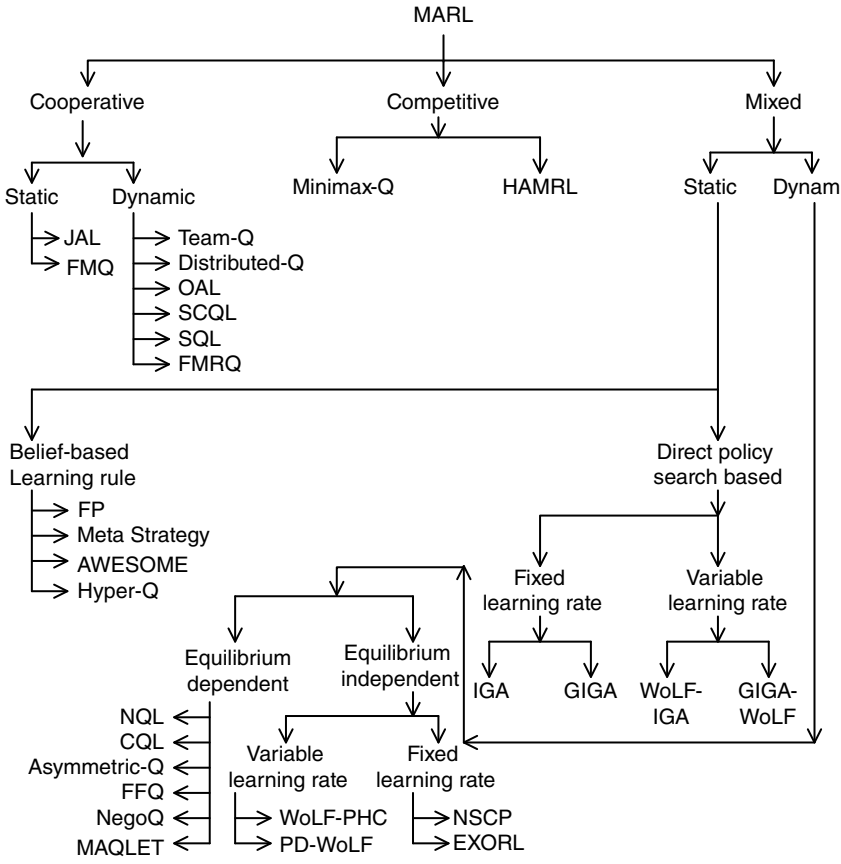


Figure 1.36 Classification of multi-agent reinforcement learning.

strategy algorithms are classified as equilibrium dependent and equilibrium independent. Equilibrium-dependent algorithms are Nash Q-Learning (NQL), Correlated Q-Learning (CQL), Asymmetric-Q learning, Friend-or-Foe Q-Learning (FFQ) (for more than two agents), Negotiation-based Q-learning (NegoQ), and MAQL with equilibrium transfer (MAQLET). Again equilibrium-independent learning algorithms are classified as fixed learning rate and variable learning rate. Fixed learning rate includes Nonstationary Converging Policies (NSCP) and Extended Optimal Response Learning (EXORL) heuristic. WoLF Policy Hill-Climbing (WoLF-PHC) and PD-WoLF are under the variable learning rate. The details of all the algorithms are given in the subsequent sections.

1.3.5.1 Cooperative MARL

The cooperative MARL algorithms are given below as listed in Figure 1.36.

Static

The static MARL does not involve any state-transitions as described in Section 1.3.3. The static MARL algorithms are discussed below.

Independent Learner and Joint Action Learner In [81], Claus and Boutilier proposed two variants of learners. One is the Independent Learner (IL) and another is the JAL. IL learns Q-value at its own action–space employing the classical single agent Q-learning rule ignoring the presence of other agents. For an IL i, the single agent Q-learning rule (1.19) becomes (1.32) with $<a, r>$ as the experience profile. Also the Q-value earned by IL denoted by $Q_i(a_i)$ converges to the optimal Q-value $Q_i^*(a_i)$ for all action $a_i \in A_i$ in the single agent system.

$$Q_i(a_i) \leftarrow Q_i(a_i) + \alpha[r_i(a_i) - Q_i(a_i)]. \tag{1.32}$$

In MAS, all the agents are adapting simultaneously and hence, the environment is no longer stationary, which does not ensure the convergence of Q-values any more. Reconsidering the 1.5, in IL, Agent X learns for the actions x_0 and x_1. However, if Agent X is a JAL, then it learns for the four joint actions. It is interesting that the expected value of selecting x_0 and x_1 exclusively depends on the strategy played by Y. In 1.5, if $a = b = 10$, then Agent X's expected Q-value for x_0 is

$$\begin{aligned}\widehat{Q}_x(x_0) &= \widehat{Q}_x(x_0,y_0) \times P(y_0 \mid x_0) + \widehat{Q}_x(x_0,y_1) \times P(y_1 \mid x_0) \\ &= 10 \times P(y_0 \mid x_0) + 0 \times P(y_1 \mid x_0) \quad \left[\text{by } 1.5\, \widehat{Q}_x(x_0,y_0) = 10 \text{ and } \widehat{Q}_x(x_0,y_1) = 0\right] \\ &= 10 \times 0.5 + 0 \times 0.5 \\ &= 5,\end{aligned} \tag{1.33}$$

where $P(y_0 \mid x_0)$ and $P(y_1 \mid x_0)$ refer to the probability of y_0 and y_1 being executed, respectively, by agent Y subject to x_0 is being selected by agent X. To handle the above explained dynamics in multi-agent systems, the JAL maintains a belief about the other agents' strategies and the expected value of action a_i by agent i is given below.

$$\hat{Q}_i(a_i) = \sum_{a_{-i} \in A_{-i}} Q_i(a_{-i}, a_i) \prod_{j \neq i} P^i_{a_{-i[j]}}, \tag{1.34}$$

where

$$Q_i(a_{-i}, a_i) \leftarrow Q_i(a_{-i}, a_i) + \alpha[r_i(a_{-i}, a_i) - Q_i(a_{-i}, a_i)]. \tag{1.35}$$

The experience tuple of JAL is denoted by $<a_i, a_{-i}, r_i>$.

So, JAL learns the Q-value at joint action–space considering the presence of other agents by synergistically combining the RL and equilibrium (or coordination) learning methods [45, 86–88]. Learning equilibrium depends on the rewards corresponding to the joint actions at a given joint state and these rewards are obtained by the well-known RL more especially by Q-learning. The convergence of Q-learning does depend on the already explained trade-off between the exploration and exploitation. If an agent i chooses an action a_i with probability $P_i(a_i)$, then probability of choosing remaining actions is $1 - P_i(a_i)$. The said trade-off can be balanced by tuning the temperature parameter T of the Boltzmann strategy given by (1.36). The variation of T is done in such a way so that the convergence is guaranteed [89].

$$P_i(a_i) = \frac{e^{Q_i(a_i)/T}}{\sum\limits_{\forall \widetilde{a_i}} e^{Q_i(\widetilde{a_i})/T}}. \tag{1.36}$$

The following conditions are required to satisfy for convergence of both the IL and JAL [81]:

1) The learning rate α decreases with respect to time, i.e. $\sum_{\alpha=0}^{t} \alpha = \infty$ and $\sum_{\alpha=0}^{1} \alpha^2 < \infty$.
2) Each agent selects each of its actions infinitely.
3) The probability of choosing action a by agent i, $P_t^i(a) \neq 0$.
4) All the agent's exploration strategy is exploitive. That is, $\lim_{t \to \infty} P_t^i(X_t) = 0$, where X_t is a random variable denoting the event that some nonoptimal action was taken based on i's estimated value at time t.

Finally, myopic heuristic-based optimistic exploration strategies are proposed in [81] for optimal action selection.

1) **Optimistic Boltzmann (OB):** Choose the action a_{-i} using the Boltzmann strategy, assuming $\text{Max}Q_i(a_i) = \text{Max}\, Q_i(a_i, a_{-i})$.
2) **Weight OB (WOB):** Explore using the Boltzmann strategy using the factor P_i(optimal match a_{-i} for a_i), i.e. $\text{Max}Q_i(a_i) \cdot P_i$(optimal match a_{-i} for a_i).
3) **Combined:** Employ the Boltzmann strategy, assuming $V(a_i) = \rho\, \text{Max}\, Q_i(a_i) + (1 - \rho)EV(a_i)$ as the value of action a_i, where $\rho \in [0, 1]$.

Considering the biasing $\rho = 0.5$ in [81], it is shown that combined exploration strategy outperforms the OB, WOB, and the Boltzmann strategy in terms of the average accumulated reward. The algorithm for IL and JAL are given in Algorithms 1.4 and 1.5, respectively.

Algorithm 1.4 Independent Learners

Input: Action set of agent i, A_i, $\alpha \in [0, 1)$;
Output: Optimal Q-value of agent i, $Q_i^*(a_i)$, $a_i \in A_i$;
Initialize: $Q_i(a_i) \leftarrow 0$;
Begin
Repeat
Execute an action a_i by agent i employing the Boltzmann strategy;
Receive immediate reward $r_i(a_i)$;
Update: $Q_i(a_i) \leftarrow Q_i(a_i) + \alpha[r_i(a_i) - Q_i(a_i)]$;
$Q_i^*(a_i) \leftarrow Q_i(a_i)$;
Until $Q_i(a_i)$ converges;
End.

Algorithm 1.5 Joint Action Learners

Input: Action set A_i, $\forall i$, $\alpha \in [0, 1)$;
Output: Optimal joint Q-value $Q_i^*(a_i, a_{-i})$, $\forall i$;
Initialize: $Q_i(a_i, a_{-i}) \leftarrow 0$;
Begin
Repeat
Execute an action a_i by agent i employing the Boltzmann strategy;
Receive immediate reward $r_i(a_i, a_{-i})$ by observing other agents' rewards;
Update: $\begin{array}{l} Q_i(a_i, a_{-i}) \leftarrow Q_i(a_i, a_{-i}) + \\ \alpha[r_i(a_i, a_{-i}) - Q_i(a_i, a_{-i})] \end{array}$ and
$P^i_{a_j}$, $\hat{Q}_i(a_i)$ by (1.50), (1.51) respectively;
$Q_i^*(a_i, a_{-i}) \leftarrow Q_i(a_i, a_{-i})$;
Until $Q_i(a_i, a_{-i})$ converges;
End.

Unfortunately, the above conditions do not guarantee convergence to equilibrium in the practical and complicated games such as in the climbing game [3, 81] and the penalty game [3, 81].

Frequency Maximum Q-Value heuristic The independent agent in [90] and [2] including the JAL in [81] does not guarantee convergence to the optimal joint action in the absence of coordination with high penalties. In the FMQ heuristic [3], a novel action selection strategy is proposed assuming agents can observe other agents' actions and are tested in two coordination problems mentioned in [81]: the climbing game and the penalty game. The said games are repeated cooperative single-stage games and they provide suitable platforms for studying the multi-agent coordination problem.

Agent 1 \ Agent 2	x	y	z
x	11	−30	0
y	−30	7	0
z	0	6	5

Figure 1.37 The climbing game reward matrix.

Climbing game: It is apparent from Figure 1.37 that in the climbing game [3, 81], (x, x) is the optimal joint action and both the agents should go for it. Now, if Agent 1 plays x and Agent 2 plays y, then both the agents receive negative reward (−30). After learning this situation, both the agents avoid joint action (x, y). Later, if Agent 1 plays action z, then Agent 2 plays either y or z as due to both the joint action (z, y) and (z, z) agents receive positive rewards of 6 and 5, respectively. Suppose, Agent 2 is playing x but Agent 1 does not play x as it receives negative reward in the past due to x, and also Agent 1 does not play y as it provides negative reward. Hence, Agent 1 plays z and both the agents receive reward of 0. Similarly, if Agent 2 plays z, then agents receive at least 0 independent of Agent 1's choice. From the above analysis it is apparent that in the climbing game, agents always move away from the optimal joint action.

Penalty game: Similar to the climbing game, the presence of multiple equilibria in the penalty game [3, 81] is also challenging to check the performance of the coordination in the MAS. In the penalty game (Figure 1.38), both the agents should avoid the joint actions (x, z) and (z, x) to avoid the negative reward of −10. Now, in the penalty game, there are two optimal joint actions (x, x) and (z, z). Agents can play for any one of them. Suppose, Agent 1 plays x with an expectation that Agent 2 also plays x to receive maximum reward of 10. In this situation, if Agent 2 plays z, expecting Agent 1 plays z to receive maximum reward of 10. In the above circumstances, y is the safe choice for both the agents regardless of what other agent's play and is guaranteed to receive a reward of 0 or 2. Hence, it is challenging to identify the optimal joint action in penalty game for multi-agent coordination.

From the climbing game and the penalty game it is apparent that an agent should select its action wisely for convergence. Maintaining a balance between the exploration and exploitation is an intelligent approach for action selection. Balancing the exploration/exploitation is a trade-off and is addressed by the well-known

Agent 1 \ Agent 2	x	y	z
x	10	0	−10
y	0	2	0
z	−10	0	10

Figure 1.38 The penalty game reward matrix.

Boltzmann strategy given in (1.36). In (1.36), the probability of selecting an action a_i for agent i is evaluated by utilizing the Q-value and the tuning parameter temperature (T). If $T \rightarrow \infty$, then each action has an equal probability to execute and hence, pure exploration occurs. If $T \rightarrow 0$, then the action has a probability of one to execute and hence, exploitation occurs. In [3], T is given by

$$T(t) = e^{-st} \times T_{\max} + 1, \tag{1.37}$$

where t is the learning epoch, s is a parameter to control the exploration rate, and $T_{\max}$ is initial value of temperature.

In [91], an optimistic assumption-based algorithm is proposed. By optimistic assumption, an agent updates its Q-value only if the new value is greater than the current one. Unfortunately, the optimistic assumption fails to converge to the optimal joint action due to misleading maximum reward. FMQ heuristic is based on the experience of the agent. Agent counts the frequency of the action which yields the best reward. Instead of optimistic assumption, an agent i uses the Boltzmann strategy with the modified Q-value $\widetilde{Q}_i(A)$ given in (1.38).

$$\widetilde{Q}_i(A) = Q_i(A) + f \times \frac{c_{\max}(A)}{c(A)} \times r_{\max}(A), \tag{1.38}$$

where $c_{\max}(A)$ is the number of times agent i receives maximum reward $r_{\max}(A)$ after executing the action A $_c(A)$ times. f refers to the control parameter to control the importance of the FMQ heuristic. The value of f increases proportionally with the increase in problem difficulty.

Algorithm 1.6 FMQ heuristic

Input: Action set A_i, $\forall i$, $\gamma \in [0, 1)$, $\alpha \in [0, 1)$, f;
Output: Optimal joint Q-value $Q_i^*(A)$, $\forall i$, $i \in [1, m]$;
Initialize: $Q_i(A) \leftarrow 0$, $\forall i$;
Begin
  Repeat
    Execute action $a_i \in A_i$, $\forall i$ employing FMQ heuristic;
    Receive immediate reward $r_i(A)$, $\forall i$;
    Update: $Q_i(A) \leftarrow Q_i(A) + \alpha[r_i(A) - Q_i(A)]$ and modify Q-value $\widetilde{Q}_i(a_i)$, $\forall i$ by (1.54) for
    modified Boltzmann strategy (FMQ heuristic);
    $Q_i^*(A) \leftarrow Q_i(A)$;
  Until $Q_i(A)$, $\forall i$ converge;
End.

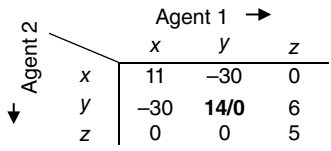


Figure 1.39 The penalty game reward matrix.

It is observed from the experiments that the FMQ heuristic outperforms the baseline experiments in terms of the convergence to the optimal joint action both in the climbing game and the penalty game [3]. To compare the FMQ heuristic with optimistic assumption, a partially stochastic version of the climbing game is given in Figure 1.39. In the partially stochastic climbing game, at least one of the rewards is stochastic as shown in Figure 1.39. In Figure 1.39, the joint action (y, y) yields a reward of 14 or 0 with probability 0.5. So, in the long run both the agents receive a reward of 7 due to joint action (y, y). Hence, the reward matrix given in Figures 1.37 and 1.39 are equivalent in the long run. The FMQ heuristic also outperforms the baseline experiment and the optimistic assumption in the partially stochastic climbing game, in terms of the convergence to optimal joint action. Unfortunately, the FMQ heuristic fails to convergence to optimal joint action in the fully stochastic penalty game and climbing game. The algorithm for FMQ heuristic is given in Algorithm 1.6.

Dynamic

Dynamic RL is stochastic Markov game with more than one joint state.

Team-Q Team-Q is a cooperative dynamic Q-learning algorithm. Dynamic indicates the existence of state-transitions. In [92], Littman proposed Team-Q learning designed for team games in the framework of team Markov games (Coordination game). In Team-Q learning, the value function $VQ_i(S^{/})$ of agent $i \in [1, m]$ at joint next state $S^{/}$ for the m agents' team is given in (1.39).

$$VQ_i\left(S^{/}\right) = \underset{a_1, a_2, ..., a_m}{\text{Max}} Q_i(S; a_1, a_2, ..., a_m). \tag{1.39}$$

The update rule in Team-Q learning for agent i is given in (1.40), without using reaming agents' model like in [81].

$$Q_i(S,A) \leftarrow (1-\alpha)Q_i(S,A) + \alpha\left[r_i(S,A) + \gamma VQ_i\left(S^{/}\right)\right], \tag{1.40}$$

where $A = < a_1, a_2, ..., a_m >$ be the joint action at joint state $S = < s_1, s_2, ..., s_m >$. The Team-Q learning is convergent following the generalized Q-learning algorithm [93, 94]. Team-Q learning is similar to NQL [95] for the coordination games. Still there exists a challenge regarding the equilibrium selection among multiple equilibria in noisy environment. The algorithm for Team-Q learning is given in Algorithm 1.7.

Algorithm 1.7 Team-Q

Input: Action set $A_i, \forall i, \gamma \in [0, 1), \alpha \in [0, 1)$;
Output: Optimal joint Q-value $Q_i^*(S, A), \forall i, i \in [1, m]$;
Initialize: $Q_i(S, A) \leftarrow 0, \forall i$;
Begin
  Repeat
    Execute action $a_i \in A_i, \forall i$;
    Receive immediate reward $r_i(S, A), \forall i$;
    Update: $Q_i(S, A) \leftarrow (1-\alpha)Q_i(S, A) + \alpha\left[r_i(S, A) + \gamma \max_{A} Q_i(S; A)\right]$
    and $S \leftarrow S'$;
    $Q_i^*(S, A) \leftarrow Q_i(S, A)$;
Until $Q_i(S, A), \forall i$ converge;
End.

Distributed-Q In [91], model-free Distributed Q-learning is proposed for cooperative MAS in deterministic situation with a motivation to compute an optimal policy in a cooperative multi-agent environment. The Distributed Q-learner solves two problems. The first problem is concerned with determination of the optimal policy. The second problem deals with selection of one optimal policy among alternatives, which is optimal for the entire team.

To handle multi-agent dynamics, MDP is extended to Multi-agent MDP (MMDP), where each agent maximizes its own reward having different goals (i.e. reward-function). However, in the cooperative MMDP, all the agents have identical reward function. Such identical reward-functions are advantageous in finding an equilibrium point, which is an optimal joint action and it maximizes the reward of all the agents. In cooperative MMDP, the learning algorithm is responsible in making cooperation among the agents. Here, also two types of agents are considered: one is JAL and another is IL as mentioned in [81]. IL cannot distinguish the difference between the individual (elementary) action [91] and joint action. Hence, IL maintains a Q-table of smaller size, i.e. $S \times A$, instead of maintaining the Q-table at joint state–action space, $S \times A^m$. In [91], the smaller Q-tables are assumed as the projection from the larger central Q-table with a conjecture about the strategies of teammates. So, in [91], a projection approach is proposed by evaluating the individual Q-table in a distributed way without adapting Q-table in joint state–action space by weighting the Q-values from larger Q-table given in (1.41).

$$q_i(S,a_i) \leftarrow \sum_{\forall A = <a_i>_{i=1}^{m}} \left[P(S,A \mid a_i) \cdot \left[r_i(S,a_i) + \gamma \max_{a_i^{/}} q_i\left(S^{/}, a_i^{/}\right) \right] \right], \tag{1.41}$$

where $P(S, A \mid a_i)$ refers to the probability of joint action A to be executed at joint state S including the action of agent i, a_i.

Another way of projection is the "pessimistic assumption." By pessimistic assumption, the individual smaller Q-value is the least efficient agent's Q-value obtained from larger central Q-value. Such approach creates robust policies but is not extended in [91], because of its cautious nature. Instead of pessimistic assumption, its dual form is utilized to obtain the smaller Q-value from the central Q-table as given in (1.42).

$$q_i(S,a_i) \leftarrow \max_{A} Q(S,A). \tag{1.42}$$

It can also be written in terms of small Q-table given in (1.43).

$$q_i(S,a_i) = \max_{a_i \in A} q_i(S,a_i). \tag{1.43}$$

Algorithm 1.8 Distributed Q-learning

Input: Action set A_i, $\forall i$, $\gamma \in [0, 1)$;
Output: Optimal Q-value $q_i^*(S, a_i)$, $\forall i$, $i \in [1, m]$, $a_i \in A$;
Initialize: $q_i(S, a_i) \leftarrow 0$, $\forall i$;
Begin
  Repeat
    Execute action $a_i \in A_i$, $\forall i$;
    Receive immediate reward $r_i(S, a_i)$, $\forall i$;
    Update: $q_i(S,a_i) \leftarrow max\left\{ q_i(S,a_i), r_i(S,a_i) + \gamma \max_{a_i^{/}} q_i\left(S^{/}, a_i^{/}\right) \right\}$
    and $S \leftarrow S^{/}$;
    $q_i^*(S, a_i) \leftarrow q_i(S, a_i)$;
  Until $q_i(S, a_i)$, $\forall i$converge;
End.

The projection technique introduced in (1.43) is also known as optimistic assumption. It is assumed that all agents are acting optimally and the conjunction of the individual optimal actions is also an optimal joint action. However, such assumption is necessarily not true. This inspires the researchers in [91] to propose a Proposition, which states that in cooperative deterministic MMDP

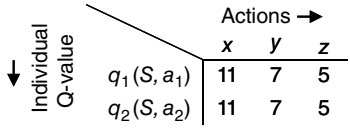


Figure 1.40 Individual Q-values obtained in the climbing game reward matrix by Distributed Q-learning.

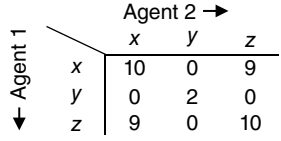


Figure 1.41 The penalty game reward matrix.

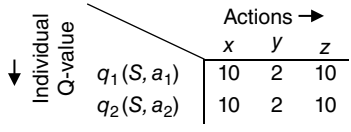


Figure 1.42 Individual Q-values obtained in the penalty game reward matrix by Distributed Q-learning.

$$q_i^t(S, a_i) = \max_{A = < a_i > \atop i=1}^{m} Q^t(S, A) \quad (1.44)$$

holds and also its proof is given in [91], where t is the learning epoch. The steps of the Distributed Q-learning are given in Algorithm 1.8. The climbing game and the penalty game are extended for the Distributed Q-learning, respectively, in Examples 1.9 and 1.10.

Example 1.9 To extend the climbing game for Distributed Q-learning as shown in Figure 1.40, let both the agents are at joint state S and for brevity discount factor γ is set to 0. The reward function $q_i(S, a_i)$ is evaluated employing Algorithm 1.8. Such greedy approach of Algorithm 1.8 yields highest Q-value for both the agents as shown in Figure 1.40. In Figure 1.40, the optimal joint action is (x, x). However, in the IL, JAL, and the FMQ-heuristic algorithm, agents are supposed to find the suboptimal joint action (y, y) as explained in Figure 1.37.

Example 1.10 Like Example 1.9 to extend the penalty game for Distributed Q-learning as shown in Figure 1.42, the discount factor γ is set to 0 and Algorithm 1.8 is employed to evaluate distributed rewards. In the FMQ-heuristic algorithm (Figure 1.41), there are four optimal joint actions:(x, x), (x, z), (z, x), and (z, z) but only (x, x) and (z, z) are optimal joint actions with reward 10 as shown in Figure 1.42 offered by Algorithm 1.8. Unfortunately, application of the Distributed Q-learning is limited to the deterministic system only.

Optimal Adaptive Learning There are many straightforward solutions to choose optimal equilibrium among multiple equilibrium solutions, like enforce convention [96] and FP [39, 81]. In [81], the JAL guarantees the convergence to NE in a team game. However, it is not guaranteed that the selected NE is the optimal one. Similar problem arises in game theory like Adaptive play (AP) [86] and evolutionary model proposed in [40].

In model-free RL, agents do not have any idea about the environment; in addition, they may receive noisy rewards. Hence, it is impossible to converge

← Agent 1 \ Joint actions of Agents 2 and 3 →	b_1c_1	b_1c_2	b_1c_3	b_2c_1	b_2c_2	b_2c_3	b_3c_1	b_3c_2	b_3c_3
a_1	**10**	–20	–20	–20	–20	5	–20	5	–20
a_2	–20	–20	5	–20	**10**	–20	5	–20	–20
a_3	–20	5	–20	5	–20	–20	–20	–20	**10**

Figure 1.43 Reward matrix of a three-player coordination game.

properly. In [91] and [96], the MDP is extended to Team Markov Game (cooperative MMDP) with an aim to find a deterministic joint strategy to maximize the expected sum of discounted rewards. In [96], the OAL algorithm is proposed with convergence proof where agents learn to choose the optimal NE among multiple NE with probability one. Let in a three-player coordination game, $<a_1, a_2>$, $<b_1, b_2>$, and $<c_1, c_2>$ be the individual action sets of agent 1, 2, and 3, respectively. The reward matrix of this coordination game is shown in Figure 1.43. It is apparent from Figure 1.43 that there are three PSNEs $<a_1b_1c_1, a_2b_2c_2, a_3b_3c_3>$ and six suboptimal NEs. The rewards corresponding to the suboptimal NE are italicized.

Before discussing about the OAL algorithm, the AP algorithm [86] is discussed. In AP game, it is assumed that agents know the game before playing it and one virtual game (VG) is designed. In Team Markov Game, to eliminate the suboptimal NE, the following arrangement is made. Suppose, in cooperative situation $VG(S, A)$ be the payoff of the agents at joint state S because of joint action A. In VG, it is assumed that at optimal NE, the reward denoted by $VG^*(S, A)$ is equal to one and else it is set to zero; e.g. in Figure 1.43, $VG^*(S, A)$ is equal to one if A is an optimal NE, i.e. $A \in \{a_1b_1c_1, a_2b_2c_2, a_3b_3c_3\}$ and else it is zero. Considering weakly acyclic game (WAG) [86] as a VG, where each joint action $A \in \{A\}$ is considered as a vertex. The vertices are connected with the directed edge avoiding self-loop, where for an agent i the action $a_i \in A_i$ is the best response to A_{-i}, here $-i$ stands for all except agent i. By the principle of *WAG* represented as a best-response graph, from any starting vertex A there exists a directed path to some vertex $A^* \in \{A\}$ and from A^* there is no outgoing path [86].

To eliminate the suboptimal NE or tackle the equilibrium selection problem in WAG, Young proposed AP in [86]. In AP, suppose in a $m-$ player matrix game the joint action at time t is denoted by $A^t \in \{A\}$. Also assume two integers k and n such that $1 \leq k \leq n$ and $t \leq n$. After acting randomly, agents look at its experience and restart the learning at $t = n + 1$. At $t = n + 1$, each agent looks reverse at their most recent n experiences and randomly choose k samples from that. Now, the expected reward of agent i ' s action a_i is given in (1.45). After evolution of $ER(a_i)$ randomly, an action is chosen from a set of best response given in (1.46).

$$ER(a_i) = \sum_{A_{-i} \in \{A_{-i}\}} u_i(\{a_i\} \cup A_{-i}) \frac{K_{t+1}(A_{-i})}{k}, \tag{1.45}$$

where $K_{t+1}(A_{-i})$ refers to the count the joint action A_{-i} in the k samples and $u_i(\{a_i\} \cup A_{-i}) = u_i(A)$ is the reward of agent i because of joint action A.

$$BR_i^t = \left\{ a_i \mid a_i = \arg\max_{a_i^{/} \in A_i} ER\left(a_i^{/}\right) \right\}. \tag{1.46}$$

It is shown in [86] that by AP, WAG converges to a strict NE. Unfortunately, all the VGs are not WAG and hence, the AP may not converge to a strict NE for all VGs. To address the said problem, the WAG and AP algorithms are modified as follows.

The WAG and AP are modified as WAG with respect to a biased set (WAGB). In WAGB, there is a set D containing a few Nash equilibria of the WAGB. A game is a WAGB if from any vertex A one path exists leading to the NE belongs to set D or a strict NE [96]. In AP, agents randomly select the NE among multiple best responses of the agents. On the other hand, in biased AP (BAP) [96], agents deterministically select the best-response action as a NE belongs to D. Suppose, W_t denotes the set of k samples drawn from the most recent n joint actions. The following two conditions are satisfied. First condition is that the joint action $A^{/} \in D$ such that $\forall A$, $A \in W_t$, $A_{-i} \subset A$, and $A_{-i} \in A^{/}$. Second condition is that there must exist at least a joint action $A \in D$ so that $A \in W_t$ and $A \in D$. If the above two conditions are satisfied, then agent i chooses its best-response action a_i such that $a_i \in a^{t'}$, where

$$t' = \max\left\{T \mid a^T \in W_t \wedge a^T \in D\right\}. \tag{1.47}$$

The philosophy of (1.47) is that the action a_i is the component of the most recent NE belonging to D. If the above two conditions are not satisfied, then AP is implemented. Hence, it can be concluded that the BAP on WAGB converges to either a NE belongs to D or a strict NE. The above techniques are applicable only when the game structure is known. To learn in an unknown game structure multi-agent $\in-$optimality is employed. By definition, a joint action is $\varepsilon-$optimality at joint state S and time t if $Q_t(S,A) + \in \geq \max_{A^{/}} Q_t\left(S,A^{/}\right), \forall A^{/} \in \{A\}$. Let the set of $\varepsilon-$optimal joint action, which converges Q_t to Q^* with slower rate, then VG_t converges to VG^*. Here, ε varies proportionately to the function $B(N_t) \in [0, 1]$, where $B(N_t)$ decreases slowly and monotonically to zero with N_t. N_t refers to the minimum time required to sample a state–action pair. The algorithm for OAL is given in Algorithm 1.9 [96]. The convergence proof of the OAL algorithm is given in [96].

Algorithm 1.9 Optimal Adaptive Learning

Input: Action set A_i, $\forall i$ at joint state S, $\gamma \in [0, 1)$;
Output: Optimal Q-value $Q^*(S, A)$;
Initialize:
$t = 0$, $n_t(S, A) = 1$, $T_t\left(S^{/} \mid (S, A)\right) = \dfrac{1}{|S|}$, $R_t(S, A) = 0$, $\in_t = C$, $A^{\in_t}(S) = A$, $D = A$;
Repeat // $n_t(S, A)$ is the number of times the joint action A has been executed in joint state S up to time t
If $t \le m$,
  Then randomly select an action a_i, $\forall i$;
Else do
  Begin
  Update the virtual game VG_t at joint state S;
  Randomly select records from n recent observations of other agents' joint actions played at joint state S;
  Evaluate expected payoff of individual action a_i of the VG at joint state S by (1.45) and construct the best response set by (1.46);
  If condition 1 and 2 in BAP are TRUE
    Then choose best response action with respect to the biased set D;
  Else randomly select a best response action from $BR_i^t(S)$;
  End If.
  End.
End If.
  Receive immediate reward $r_i^t(S, A)$;
  Update: $n_t(S, A) \leftarrow n_t(S, A) + 1$, $R_t(S, A) \leftarrow R_t(S, A) + \frac{1}{n_t(S, A)}\left(r_i^t(S, A) - R_t(S, A)\right)$,
  $T_t\left(S^{/} \mid (S, A)\right) \leftarrow T_t\left(S^{/} \mid (S, A)\right) + \frac{1}{n_t(S, A)}\left(1 - T_t\left(S^{/} \mid (S, A)\right)\right)$,
$Q_{t+1}(S, A) \leftarrow R_t(S, A) + \gamma \sum_{\forall S^{/} \in \{S\}} T_t\left(S^{/} \mid (S, A)\right) \times \max_{\forall A^{/} \in \{A\}} Q_t\left(S^{/}, A^{/}\right)$, $t \leftarrow t + 1$, $N_t \leftarrow \min_{S, A} n_t(S, A)$;
If $\in_t > CB(N_t)$
  Then do
    Begin
    $\in_t > CB(N_t)$, $Q^*(S, A) \leftarrow Q(S, A)$, $\forall i$ and $A^{\in_t}(S)$
    $\leftarrow \left\{ A \mid Q_t(S, A) + \in_t \ge \max_{A^{/} \in \{A\}} Q_t\left(S, A^{/}\right) \right\}$;
    End;
  End If.
Until $Q(S, A)$, $\forall i$ converge;

Sparse Cooperative Q-learning One of the principal bottlenecks of the MAS is the exponential increase in the space and time complexity, with the increase in number of agents. Kok et al. [97] observed that in most of the MAS, agents are required to coordinate their actions only in a few states and in the remaining, they act independently. In the coordinated joint state S, the Q-value of an agent i is denoted by $Q_i(S, A)$. However, if S be the uncoordinated joint state, then the Q-value of agent i is denoted by $Q_i(S, a_i)$. In case of uncoordinated joint state, the global Q-value $Q(S, A)$ for m number of agents is defined as the summation of individual Q-values given by (1.48).

$$Q(S,A) = \sum_{i=1}^{m} Q_i(S, a_i). \tag{1.48}$$

Based on the above observations, in [97], Kok and Vlassis proposed SCQL, where the Q-tables of the agents are sparsely maintained as discussed above.

Sequential Q-learning In [98], Wang and Silva proposed SQL to handle conflicting behavior of the agents that arises in tightly coupled multi-robot object transportation. In SQL, robots do not select their actions simultaneously; rather they do it sequentially based on their predefined priorities. In SQL, the problem of behavior conflict is addressed by avoiding the selection of same actions those already selected by the preceding robots. Assuming ith robot is denoted by R_i, $i \in [1, m]$ and all the robots are arranged in a special sequence. The subscript i in R_i indicates its position in the sequence. All the robots repeat steps given in Algorithm 1.10 to

Algorithm 1.10 Joint Action Formation in SQL

Initialize $\Psi = \phi$; // ϕ be the empty set.
Observe current joint state S;
For $i = 1$ to m
  Evaluate the currently available action set Δ_i, where A_i the action set of robot be i.
  $\Delta_i = (A_i - (A_i \cap \Psi))$;
  R_i selects the action $a_i^j = \Delta_i$ by probability $P(a_i^j) = \dfrac{e^{Q_i(S, a_i^j)}}{\sum_{r=1}^{|\{\Delta i\}|} e^{Q_i(S, a_i^r)}}$.
  Include the action a_i^j to the set Ψ;
End For
Execute the corresponding selected action $a_i^j, \forall i$;

form a joint action avoiding the conventional steps in the classical step MAQL. The joint action offered by Algorithm 1.10 avoids the bottleneck of behavior conflict in tightly coupled multi-robot object transportation.

Frequency of the Maximum Reward Q-learning In [99], Zhang et al. proposed a MARL algorithm for fully cooperative tasks, namely FMRQ, which aims at achieving the optimal NE to maximize the system performance with respect to the metric of interest. In FMRQ, a modified immediate reward signal is used, which is obtained by identifying the highest global immediate reward. In FMRQ, an agent needs to share only its state and reward at each learning epoch with remaining agents.

In FMRQ, the authors considered two issues: first, they investigated whether the NE is good enough for the fully cooperative MAS; second, the curse of dimensionality of the MARL is considered by storing the Q-value at joint state–individual action space.

To describe the dynamics of the FMRQ, differential equations are formulated for the four cases including two-agent two-action repeated game, and a three-agent two-action repeated game. In each case, the critical points of the differential equations are analyzed and it is observed that FMRQ converges to equilibrium with maximum global rewards in all the five cases [99]. In case 1, there exists only one global immediate reward. Cases 2 and 3 have two maximum immediate rewards in diagonal positions and in the same row, respectively. In case 4, three maximum immediate rewards exist and in case 5, only one global immediate reward exists [99].

In FMRQ, the size of a Q-table for an agent i is $|\{S\}| \times |\{A_i\}|$. In the FMRQ algorithm (Algorithm 1.11), the immediate reward of an agent i, denoted by $r_i(a_i)$, is replaced by the frequency of getting the maximum global immediate reward by the same action a_i, denoted by $fre(a_i)$.

$$fre(a_i) = \frac{n_{\max_{a_i}}}{n_{a_i}}, \tag{1.49}$$

where n_{a_i} refers to the number of times action a_i is selected by agent i and $n_{\max\_a_i}$ is the number of times agent i achieves the maximum global immediate reward. Moreover, the superiority of the FMRQ algorithm is verified by two case studies: one is the 12-vertex box-pushing by 4-agents and the other one is the distributed sensor network optimization problem. The FMRQ algorithm is provided in Algorithm 1.11 for an agent i in repeated games.

Algorithm 1.11 FMRQ for an Agent *i* in Repeated Games

Input: Action set $a_i \in A_i$, $\forall\, i$ and learning rate $\alpha \in [0, 1)$;
Output: Optimal Q-value $Q^*(a_i)$;
Initialize: $Q(a_i) \leftarrow 0$, count of selecting action a_i, $n_{a_i} = 0$, number of times maximum global immediate reward received by action a_i, $n_{\max\_a_i} = 0$ and frequency of getting maximum immediate reward after selecting action a_i, $fre(a_i) = 0$;
Repeat
  Select an action a_i by the Boltzmann exploration scheme;
  $n_{a_i} = n_{a_i} + 1$;
  Execute the action a_i and update $n_{\max\_a_i}$ and $rh(a_i)$;
  For each action $a_i \in A_i$ **do** //$rh(a_i)$ refers to history of global immediate reward obtained by action a_i
   Begin
    Evaluate $fre(a_i)$ by (1.49);
    $Q(a_i) = Q(a_i) + \alpha(fre(a_i) - Q(a_i))$;
    Set $n_{a_i} = 0$, $n_{\max\_a_i} = 0$ and $fre(a_i) = 0$;
   End
  End For
Until $Q^*(a_i)$; converges;

1.3.5.2 Competitive MARL

The competitive MARL algorithms are discussed below. Here, two competitive MARL algorithms are discussed. One is Minimax Q-learning for two agents and its extension for general-sum game for more than two agents called HAMRL.

Minimax-Q Learning

In [100], Littman proposed a competitive algorithm, namely minimax-Q learning for two agents. In minimax-Q learning, both the agents have conflicting goals with an objective of maximizing the sum of its own discounted expected reward. In other words, an agent tries to maximize a reward function and simultaneously the opponent agent tries to minimize it. In [2] and [90], the authors realized that an agent must interact with other agents and the environment during the learning phase without proposing any supporting mathematical model. In addition, the theory of MDP [46, 84], which is an extension of game theory, also cannot handle the multi-agent dynamics. Even sometimes it is assumed that the environment is stationary. Littman [100] considered only two-player zero-sum Markov game. In zero-sum game, the summation of the rewards of two agents is zero [41]. In every

MDP, there is at least one strategy that is stationary, deterministic, and optimal [100]. But in most of the cases, the optimal strategies are probabilistic. For example, in Figure 1.23 (rock, paper, and scissor, Example 1.3), selection of a deterministic policy by any one player leads to punishment and hence, the player is defeated. The probabilistic strategy is required to represent the uncertainty about the agents' action choice. Suppose, the opponent agent has an action $O \in \{O\}$ and Q-value is denoted by $Q(S, A, O)$ as introduced in (1.50).

$$Q(S,A,O) \leftarrow r(S,A,O) + \gamma \sum_{S'} P\left(S' \mid (S,A)\right) \times V\left(S'\right), \tag{1.50}$$

where

$$V\left(S'\right) = \max_{\pi \in P(\{A\})} \min_{O \in \{O\}} \pi_A \cdot Q(S,A,O). \tag{1.51}$$

(1.51) indicates the expected reward to the agent for playing strategy π against the opponent's choice $O \in \{O\}$. $P(\{A\})$ refers to the probability distribution over the action set $\{A\}$. The algorithm for Minimax Q-learning is given in Algorithm 1.12 [100]. Algorithm 1.12 is tested in a two-player Markov game and it is compared with Q-learning. The convergence of Minimax-Q learning is guaranteed and the strategy offered by it is a safe choice against the opponent even in the worst situation.

Algorithm 1.12 Minimax Q-learning

Input: Action $A \in \{A\}$, opponent's action $O \in \{O\}$ at joint state S, $\alpha \in [0, 1)$ and $\gamma \in [0, 1)$;
Output: Optimal Q-value $Q^*(S, A, O)$;
Initialize: $Q(S, A, O) \leftarrow 0$, $\pi(S, A) = \frac{1}{|A|}$;
Begin
  Repeat
    Choose an action to execute by $\pi(S, A)$;
    Receive immediate reward $r(S, A, O)$;
    Update: $Q(S, A, O) \leftarrow (1-\alpha) Q(S, A, O) + \alpha [r(S, A, O) + \gamma V(S')]$,
    $S \leftarrow S'$,
  $\pi(S, A) = \arg\max_{\pi(S,A)} \left[Min_{O} \sum_{\forall A} \pi(S, A) \times Q(S, A, O) \right]$ and $V(S') = \max_{\pi \in P(\{A\})} \min_{O \in \{O\}} \pi_A . Q(S, A, O)$;
    $Q^*(S, A, O) \leftarrow Q(S, A, O)$;
  Until $Q(S, A, O)$ converges;
End.

Heuristically Accelerated Multi-agent Reinforcement Learning

In [101], Bianchi et al. proposed HAMRL, which attempts to speed up in convergence of MARL, by balancing exploration/exploitation employing a heuristic function for action selection. There exist a series of literature [101–104], where heuristic functions are used to increase the convergence speed of the MARL. The work of [101] is the extension of [104], whereas in [104], Littman's Minimax-Q is heuristically accelerated. Bianchi et al. defined a heuristic function $H : \{S\} \times \{A\} \times \{O\} \to \mathbb{R}$, which influences the action selection of the agents during the learning phase, when an agent executes an action $A \in \{A\}$ at state $S \in \{S\}$ against the opponent's action $O \in \{O\}$. In [101], the authors employ the modified ε-greedy learning rule including the heuristic function $H(S, A, O)$ given by (1.52).

$$\pi^c(S) = \arg\max_A \min_O \left[Q(S,A,O) + \xi H(S,A,O)^\beta\right] \tag{1.52}$$

and $\xi \in \mathbb{R}$, $\beta \in \mathbb{R}$ are the weightage on the confidence of the heuristic function. In (1.52), if $\xi = 0$, then (1.52) becomes (1.53), which is the standard ε-greedy.

$$\pi(S) = \begin{cases} \pi^c(S), & \text{if} \quad p \geq \varepsilon, \varepsilon \in [0,1] \\ \text{Select an action randomly,} & \text{otherwise} \end{cases}, \tag{1.53}$$

where $p \in [0, 1]$ is a random number. Considering $\xi = \beta = 1$, the heuristic function is given by

$$H(S,A,O) = \begin{cases} \max_i Q(S,i,O) - Q(S,A,O) + \eta, & \text{if } A = \pi^H(S) \\ 0, & \text{otherwise} \end{cases}, \tag{1.54}$$

where $\eta \in \mathbb{R}$, $\pi^H(S)$ is the heuristic policy. The superiority of the HAMRL (Algorithm 1.13) is validated by conducting the experiments in two robots soccer game.

Algorithm 1.13 HAMRL for Zero-sum Game

Input: Action $A \in \{A\}$, opponent's action $O \in \{O\}$ at joint state S, $\alpha \in [0, 1)$, $\gamma \in [0, 1)$ and $\varepsilon \in [0, 1)$;
Output: Optimal Q-value $Q^*(S, A, O)$;
Initialize: $Q(S, A, O) \leftarrow 0$, $H(S, A, O)$, π^H, η;
Begin
  Repeat
    Choose an action $A \in \{A\}$ using the modified ε-greedy rule;
    Execute $A \in \{A\}$, observe the opponent's action $O \in \{O\}$;
    Receive immediate reward $r(S, A, O)$;
    Update: $Q(S, A, O) \leftarrow (1 - \alpha) Q(S, A, O) + \alpha[r(S, A, O) + \gamma V(S')]$,
    $S \leftarrow S'$ and $H(S, A, O)$,
    where $V(S') = \max_{A \in \{A\}} \min_{O \in \{O\}} Q(S, A, O)$;
    $Q^*(S, A, O) \leftarrow Q(S, A, O)$;
    Until $Q(S, A, O)$ converges;
  End.

1.3.5.3 Mixed MARL

Mixed MARL includes the following algorithms. The mixed MARL may be cooperative or competitive. It can be categorized based on the number of joint states involved: static and dynamic.

Static

The static MARL algorithms are further extended in Figure 1.36.

Belief-Based Learning Rule In belief-based learning algorithm, an agent maintains a belief about the remaining agents' strategy. This section illustrates the belief-based learning rule.

Fictitious Play FP [105] is a belief-based learning rule. Here, belief indicates that a player adapts with the strategy about opponent players' and behaves as per the strategy learned. In FP, a robot can resolve the problem of equilibrium selection [38] in a coordinated game by repeatedly playing the game by the same robot. FP is an effective and efficient approach to reach equilibrium in a coordinated game. As per FP, agent i learns the models of all the other agents$j \neq i$ by the model given in (1.55).

$$P^i_{a_j} = \frac{C^j_{a_j}}{\sum\limits_{\forall a_j} C^j_{a_j}}, \tag{1.55}$$

where $P^i_{a_j}$ refers to the model of agent j' s strategy evaluated by agent i or agent i' s assumption of playing $a_j \in A_j$ by agent j or Π_{-i} and $C^j_{a_j}$ be the number of times agent i observed agent j executing action a_j. In cooperative games, the strategy offered by (1.55) leads to an equilibrium, where in case of multiple equilibrium, agents randomly choose any one. Also, in FP, a player does not need to learn about opponent players' reward, rather it maintains a belief about the opponents' feature strategy. If a FP converges to Π^*, then Π^* is a NE.

Meta Strategy In [106], Powers and Shoham proposed a straightforward MARL algorithm for repeated games, which have the following two requirements. The first requirement is to specify a class of opponents and against them the algorithm yields a reward that approaches the reward corresponding to the best response. Second requirement is that the reward offered by the algorithm fulfills a threshold of security-level reward. Constraining the above requirements, the algorithm achieves a close to optimal payoff in self-play. Based on the above conditions an algorithm is proposed in [106], for stationary opponents only. However, to learn in a repeated game a learning algorithm is required. In the learning algorithm, an agent plays its best response with a prior probability of its opponent's strategy. GAMUT [60] is employed to test the superiority of the proposed algorithm in [106].

In [107], two properties are presented related to the rationality and convergence. By rationality in a stage game, if the other players' strategies converge to stationary strategy, then the learning algorithm will converge to a stationary strategy and it is the best response to the other players' strategies. Another property is related to the convergence. By this property, the learner will necessarily converge to a stationary strategy.

In [107], Bowling and Veloso proposed an algorithm for known repeated game having two players and two actions. Conitzer and Sandholm in [108] extend the work in [107] for all repeated games. It is investigated in [106] that the algorithms considering self-play proposed in [107] and [108] are not convergent against all possible opponents. In Figures 1.30, 1.31, and 1.34, by Tit-for-Tat algorithm for the Prisoner's Dilemma and game of chicken offers higher average reward in self-play than the rewards at NE. To avoid encounter the opponent outside the target set, security value V_s is defined in (1.56).

$$V_s = \max_{\pi_1 \in \{\pi_1\}} \max_{\pi_2 \in \{\pi_2\}} V_e(\pi_1, \pi_2). \tag{1.56}$$

In summary, Powers and Shoham synergistically fuse the FP [39], Bully [109], and Minimax [100] strategy with an aim to create most powerful hybrid algorithm [106].

By FP, an agent plays best response against its stationary opponent utilizing the likelihood of other agents to select an action from history. In [106], the best response

$$B_r(\pi) \leftarrow \operatorname*{argmax}_{x \in X}(OV_e(x, \pi)), \tag{1.57}$$

where

$$X = \left\{ y \in \Pi_1 : EV(y, \pi) \geq \max_{z \in \pi_1} (EV(z, \pi)) - \varepsilon \right\}. \tag{1.58}$$

In [106], Bully algorithm (Algorithm 1.14) is extended to handle multiple strategies with equal reward by maximizing opponent's values. In Bully algorithm, a full set of mixed strategies are

$$\mathit{BullyMixed} \leftarrow \operatorname*{argmax}_{x \in X}(OV_e(x, B_r(x))), \tag{1.59}$$

$$X = \left\{ y \in \Pi_1 : V_e(y, B_{r0}(y)) = \max_{z \in \Pi_1} (V_e(z, B_{r0}(z))) \right\}. \tag{1.60}$$

Bully algorithm is the one which is employed to elect a coordinator dynamically among m number of agents with unique identify (ID) in the field of distributed

Algorithm 1.14 Bully Algorithm

```
Begin
  An agent i initiates an election;
  Agent i sends election message to all agents with higher IDs and
  waits for feedback;
  If feedback is not OK
    Then agent i becomes coordinator and sends coordination
    message to all agents with lower IDs;
  Else
    The agent i drops out and waits for a coordination message;
  End;
  If an agent receives an election message
    Then immediately sends coordination message subject to that
    the agent has highest ID;
  Else
    Return OK and starts an election;
  If an agent receives a coordination message
    Then the agent i treats the sender as the coordination;
End.
```

computing. In distributed artificial intelligence, an algorithm needs to act as a leader (or coordinator). In distributed algorithm, it is assumed that each agent has a unique ID and goal of the algorithm is to find out the agent with highest ID. The Bully algorithm is given in Algorithm 1.14. Finally, the Minimax strategy is defined as

$$\text{maximin} \leftarrow \underset{\pi_1 \in \Pi_1}{\text{argmax}} \min_{\pi_2} V_e(\pi_1, \pi_2). \tag{1.61}$$

Initial portion of Algorithm 1.15 is related to coordination/exploration to identify the class of opponent and choose one strategy among three. If neither stationary strategy nor Bully strategy holds, then best-response strategy is applied. The algorithm plays with one of the three strategies maintaining the average reward within the security level and improving the maximum strategy when it is too low, where $d_{t_1}^{t_2}$ refers to the distribution of opponent actions for the period from t_1 to t_2. Avg_n represents the average value achieved by the agent during the last n epoch. V_{Bully} represents $V_e(BullyMixed, B_{r0}(BullyMixed))$.

Algorithm 1.15 Meta Strategy Algorithm

Begin
Set strategy = BullyMixed
  Play strategy at time step t_1;
  Play strategy at time step t_2;
  If strategy=BullyMixed AND $AVGValue_H < V_{Bully} - \varepsilon_1$ with probability P
    Then set strategy = $B_{r\varepsilon_2}\left(d_0^t\right)$ and play;
  End If
  If $\left\|d_O^{t_1} - d_{t-t_1}^t\right\| < \varepsilon_3$
    Set best Strategy = $B_{r\varepsilon_2}\left(d_0^t\right)$;
  Else if strategy=BullyMixed AND $AVGValue_H > V_{Bully} - \varepsilon_1$
    Then set Best strategy=BullyMixed;
  Else
    Set best Strategy=Best Response;
  End If
  Until end of the game;
  If $AVGValue_{t-t0} < V_{security} - \varepsilon_0$
    Play Maximin strategy for t_3 time steps
  Else
    Play best Strategy for t_3 time steps;
  End If
End

Adapt When Everybody Is Stationary, Otherwise Move to Equilibrium As per [108], the minimum requirements of MAS are that agents learn optimally against stationary opponents and converge to a NE when all the agents are playing the same algorithm. WoLF-IGA [107] has satisfied the above criteria in a two-agent two-action repeated game assuming that the opponents' strategies are observable. In [108], Conitzer and Sandholm proposed *Adapt When Everybody is Stationary, Otherwise Move to Equilibrium* (AWESOME), which is guaranteed to have the above properties for more than two agents and actions assuming that the opponents' actions (not strategies) are observable. In AWESOME, either agents' aim at adapting with the present strategies of the opponent agents or they converge to an already learned NE. Once both of the above hypotheses are discarded, agents restart the learning by the AWESOME algorithm.

The basic idea of the AWESOME is straightforward. If other agents are following stationary strategies, then AWESOME offers its best strategy to the other agents.

On the other hand, if other agents adapt their strategies, then AWESOME follows an already learned equilibrium. In spite of the above basic idea, the following additional specifications are made before proposing the AWESOME algorithm.

- From the beginning it is specified which equilibrium to repeat and restart learning by the AWESOME to avoid confusion.
- After restarting, the learning agents forget whatever it learned for simplicity.
- Following one equilibrium strategy among the already computed other equilibrium strategies may lead to divergence from equilibrium. Although, a null hypothesis exists, AWESOME does not reject the hypothesis without sufficient confirmations.
- If an agent selects its own action by its own mixed strategy, then AWESOME rejects the equilibrium strategy to avoid the nonequilibrium strategy.
- After rejecting the equilibrium strategy by AWESOME, randomly an action is chosen from a pool and changes its strategy.
- In AWESOME, except actions the strategies of the remaining agents are not observable. Hence, one needs to specify how to reject an equilibrium strategy which is common to all the agents.

The AWESOME algorithm is given in Algorithm 1.16 [108], and is developed based on the above specifications. It is shown in [108] that AWESOME learns best responses against the stationary opponents, and AWESOME converges to NE in self-play.

Hyper-Q Q-learning is a well-known technique to learn optimal strategies by an agent utilizing the cumulative rewards earned by it in an infinite trial-and-error. Unfortunately, this is not applicable for nonstationary environment with multiple adaptive agents. Most of the multi-agent Q-learner [72, 95, 100] requires knowledge about other agents' rewards and Q-function at each learning epoch. These MAQL algorithms are convergent subject to the following conditions which are not realizable in practice. First, an agent can observe all agents' rewards. Second, all the learning agents follow the same learning algorithms. In [110], Gerald proposed Hyper-Q learning. Hyper-Q learner learns only the mixed strategies and the strategies of the remaining agents are estimated employing the Bayesian inference. Hyper-Q learner aims at overcoming the above limitations of MAS by modeling the environment as repeated stochastic game, where only the remaining agents' actions are observable but the rewards received due to the actions are not observable.

Assuming the Hyper-Q learner is playing in a stochastic Markov game and hence, the reward functions of the agents become the function the available joint actions. Now, instead of choosing the best joint action with probability one (pure

Algorithm 1.16 AWESOME Algorithm

For i = 1 to m
  $\pi_i^* \leftarrow ComEquStrategy(i)$; //compute equilibrium strategy for agent i
End For;
Repeat
  For i = 1 to m
    $Ini2Empty(h_i^{prev})$; $Ini2Empty(h_i^{curr})$;
  End For;
$APPE \leftarrow true$;// All players playing Equilibrium
$APS \leftarrow true$; // All players stationary
$\beta \leftarrow false$;// β is true if the equilibrium hypothesis is just rejected
$t \leftarrow 0$; // denotes the tth epoch and is initializes to zero in every restart.
$\phi \leftarrow \pi_{Me}^*$; // refers to the AWESOME player's current strategy
While APPE
  For j = 1 to N^t
    $Play(\phi)$; //Play the strategy ϕ
    For i = 1 to m
      $Update(h_i^{curr})$;
    End For;
  End For;
  If APPE=false
    If β =false
      For i = 1 to m
        If $(\|h_i^{cur} - h_i^{prev}\| > \varepsilon_s^t)$
          Then $APS \leftarrow false$;
        End If;
      End For;
    End If;
    Then $\beta \leftarrow false$; $a \leftarrow \arg\max_a V(a, h_{-Me}^{curr})$;
    If $V(a, h_{-Me}^{curr}) > V(\phi, h_{-Me}^{cur}) + n \mid A \mid \varepsilon_s^{t+1}\mu$;
      Then $\phi \leftarrow a$;
    End If;
  End If;
  If APPE=true
    For i = 1 to m
      If $(\|h_i^{cur} - \pi_i^p\| > \varepsilon_e^t)$
        Then $APS \leftarrow false$; $\phi \leftarrow RandAct()$; $\beta \leftarrow true$;
      End If;
    End For; **End If**;
  For i = 1 to m
    $h_i^{cur} \leftarrow h_i^{prev}$; $Ini2Empty(h_i^{curr})$;
  End For; $t \leftarrow t + 1$;
End While;

strategy), in stochastic Markov game, an agent chooses actions with the best probability (mixed strategy). The Hyper-Q learning update rule is given in (1.62).

$$\Delta Q(S, p_i, p_{-i}) \leftarrow \alpha \left[r(S, p_i, p_{-i}) + \gamma \max_{a_i^{/}} Q\left(S, p_i^{/}, p_{-i}^{/}\right) - Q(S, p_i, p_{-i}) \right], \tag{1.62}$$

where p_i and $p_i^{/}$ denote the mixed strategy to select action a_i and $a_i^{/}$ at joint state S and joint next state $S^{/}$, respectively. p_{-i} and $p_{-i}^{/}$ refer to the joint mixed strategy of all the agents except i to select joint action A_{-i} and $A_{-i}^{/}$ at joint state S and joint next state $S^{/}$, respectively. It is indicated in [110] that establishing the convergence for the function, approximation-based Q-learning is more difficult than the same for the Q-learning. If all the agents do explore in a similar exploration strategy, then like Q-learning in Hyper-Q learning, agents may fail to spot the optimal mixed strategy in the strategy space after infinite visit of the joint states. In case of stationary opponent strategy, the stochastic game becomes a MDP with stationary state-transitions and stationary rewards. Under the above circumstance, Hyper-Q learning converges. Remaining convergence conditions are given in [110]. To estimate opponent strategy, Bayesian strategy estimation is done in [110]. By Bayesian estimation, one can write

$$P(S \mid H) = \frac{P(H \mid S)P(S)}{\sum_{S^{/}} P\left(H \mid S^{/}\right) P\left(S^{/}\right)}, \tag{1.63}$$

where H refers to the history of observed actions, S and $S^{/}$ are the discrete state and next state, respectively. The outstanding performance of Hyper-Q learning in terms of convergence rate and opponent agent's strategy modeling is tested in the framework of two-player, three-action matrix game rock-paper-scissors game (Example 1.3).

Direct Policy Search Based Algorithm Direct policy search-based algorithms are further classified as fixed learning rate and variable learning rate as shown in Figure 1.36.

Fixed Learning Rate The algorithms with fixed learning rates are given below.

Infinitesimal Gradient Ascent In [111], Singh and Kearns proposed IGA based on the positive changes in expected reward of the agents. The IGA is tested in a two-player, two-action iterated general-sum games. It is shown in [111] that agents converge to NE, but once they fail to converge to NE, they can never reach the

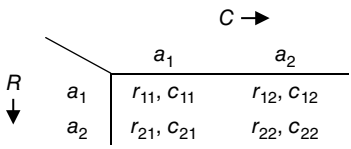


Figure 1.44 Reward matrix in a two-player two-agent game.

NE. Literature shows that agents converge to NE, but with restriction and limiting the applicability of the NE [88]. Following the gradient ascent (positive change) is the most common trend in machine learning algorithm. It is not guaranteed that the strategies computed by gradient ascent in two-player, two-action iterated games will converge to NE. However, the average reward is guaranteed to converge NE. For example, let there is a two-player, two-action general-sum game. The reward matrix of the row (R) and column (C) player is given in Figure 1.44.

Let row player choose action a_1 stochastically with probability $0 \leq r \leq 1$ and column player choose action a_1 stochastically with probability $0 \leq c \leq 1$. The expected payoff of the row and column player is given by (1.64) and (1.65), respectively.

$$V_R(r,c) = r_{11}(rc) + r_{22}((1-r)(1-c)) + r_{12}(r(1-c)) + r_{21}((1-r)c), \tag{1.64}$$

$$V_C(r,c) = c_{11}(rc) + c_{22}((1-r)(1-c)) + c_{12}(r(1-c)) + c_{21}((1-r)c). \tag{1.65}$$

Here, the strategy pair (r, c) is called NE if and only if, the following two conditions hold.

- if for any mixed strategy $r^{/}$ (1.66) holds: i.e.

$$V_R\left(r^{/}, c\right) \leq V_R(r,c) \tag{1.66}$$

- for any mixed strategy $c^{/}$ (1.67) holds: i.e.

$$V_C\left(r, c^{/}\right) \leq V_R(r,c). \tag{1.67}$$

Gradient for the row player and column player is given by (1.66) and (1.69), respectively, considering $u = (r_{11} + r_{22}) - (r_{21} + r_{12})$ and $u^{/} = (c_{11} + c_{22}) - (c_{21} + c_{12})$.

$$\frac{\delta V_R(r,c)}{\delta r} = cu - (r_{22} - r_{12}), \tag{1.68}$$

$$\frac{\delta V_C(r,c)}{\delta c} = ru^{/} - (c_{22} - c_{12}). \tag{1.69}$$

The mixed strategy update rules are given by (1.70) and (1.71), where η refers to the step size.

$$r \leftarrow r + \eta \frac{\delta V_R(r,c)}{\delta r}, \tag{1.70}$$

$$c \leftarrow c + \eta \frac{\delta V_C(r,c)}{\delta c}. \tag{1.71}$$

Assuming that the gradient ascent algorithm is a full information game and hence, both the players know the game matrices and the mixed strategies played by the opponent players in the previous step.

By game theory [43], the sequences of strategies over time may never converge to NE. However, in [111], it is shown that the average rewards of both the players always converge. The basic logic behind the analysis of two players acting according to IGA is a two-dimensional dynamic system. Considering the infinitesimal step size of $\eta(\eta \to 0)$, IGA is proposed in [111]. By (1.66)–(1.71) and setting $\eta \to 0$, the unconstraint dynamics of the strategy pair can be expressed as a function of time in (1.72).

$$\begin{bmatrix} \frac{\delta r}{\delta t} \\ \frac{\delta c}{\delta t} \end{bmatrix} = \begin{bmatrix} 0 & u \\ u^{/} & 0 \end{bmatrix} \begin{bmatrix} r \\ c \end{bmatrix} + \begin{bmatrix} -(r_{22} - r_{12}) \\ -(c_{22} - c_{21}) \end{bmatrix}. \tag{1.72}$$

If the matrix U given in (1.73) is invertible, then trajectories of the unconstraint strategies of the two-player two-action stochastic game are of having either limit cycle behavior or have divergent nature. The direction and structure of these trajectories depend on the exact values of u and $u^{/}$.

$$U = \begin{bmatrix} 0 & u \\ u^{/} & 0 \end{bmatrix}. \tag{1.73}$$

By solving (1.72), (r^*, c^*) is given in (1.74).

$$(r^*, c^*) = \left[\frac{c_{22} - c_{21}}{u^{/}}, \frac{r_{22} - r_{12}}{u}\right]. \tag{1.74}$$

The average expected reward of the IGA player converges to a NE following one of the following conditions. First condition is that the trajectories of the strategy pair will automatically converge to a NE. The other condition is that the trajectories due to the strategy pair will not converge but the average reward of the two players reward will converge to the NE. To prove these conditions, the following exclusive and exhaustive cases are considered [112].

1) U is non-invertible, if $u \neq 0 / u^{/} \neq 0$ or $u \neq 0$, $u^{/} \neq 0$. Such cases can appear in team, zero-sum, and general-sum games.
2) U is invertible, if the Eigen values of (1.75) are imaginary with zero real part, i.e. when $uu^{/} < 0$.

$$\begin{bmatrix} 0 & u \\ u^{/} & 0 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \lambda \begin{bmatrix} x \\ y \end{bmatrix}. \tag{1.75}$$

3) U is invertible, if its Eigen values are real with zero imaginary part. This condition may appear in team and general-sum games but not is zero-sum games, i.e. when $uu^{/} > 0$.

If U has imaginary Eigen values with zero real part, then based on the location of the center (i.e. (r^*, c^*)) in the two-dimensional plane, there are three possibilities.

1) The center (r^*, c^*) is in the interior of the unit square,
2) Center (r^*, c^*) is on the boundary of the unit square, and
3) Center (r^*, c^*) is outside of the unit square.

Generalized Infinitesimal Gradient Ascent Convex programming is the generalization of the linear programming having several applications in machine learning domain [113–115]. The convex programming aims at searching a point F which maximizes the cost function.

$$c : F \rightarrow \mathrm{R}. \tag{1.76}$$

A convex programming comprises a feasible set $F \subseteq R^n$ and a convex cost function given in (1.76). In applications like industrial optimization, nonlinear facility location problems [113], network routing problems [116], and consumer optimization problems [117], the value of the end product is unknown until the end product is created. In [118], an online convex optimization programming is undertaken with identical feasible set but having dissimilar cost functions. An algorithm is proposed in [118], namely GIGA, which is generally reliable to solve former problems. GIGA is the extension of IGA [111] applicable for more than two agents. Following the definitions of convex, convex programming problem, online convex programming problem, and the assumptions made in [107], make clear idea about the online convex optimization. Interestingly, it is shown in [107] that the repeated games are online linear programming. Finally, GIGA tries to minimize regret [118].

Variable Learning Rate The algorithms with variable learning rates are given below.

Win or Learn Fast-IGA Referring from Section "Infinitesimal Gradient Ascent," if the center (r^*, c^*) is inside the unit square with imaginary Eigen values, then the performance of IGA and WoLF-IGA differs in terms of convergence. It is shown in [107] that IGA does not converge if (r^*, c^*) lies inside the unit square. But WoLF-IGA converges in such situation. The strategy-space where the player wins and loses is also indicated in the proof. In addition, it is shown in [107] that the trajectories due to Eigen values are pricewise elliptical in nature and take a spiral shapes

toward the center. In [107], lemmas are proposed assuming that there are only imaginary Eigen values.

By lemma 6 in [107], if the learning rate for the row player (α_r) and the learning rate for the column player (α_c) remain constant, then the trajectory due to strategy pair forms an ellipse considering (r^*, c^*) as the center and $\begin{bmatrix} 0 \\ \sqrt{\frac{\alpha_c \mid u \mid}{\alpha_r \mid u' \mid}} \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ are as the axes of the ellipse. In [107], lemma 7 concludes that a player is winning if the strategy of the player is moving away from the center. It is also mentioned in [107] that in a two-person, two-action iterated general-sum game, both the players follow the WoLF-IGA algorithm with learning rates $\alpha_{\max}$ and $\alpha_{\min}$, then their strategies will converge to a NE subject to

$$\frac{\alpha^r_{\min}\alpha^c_{\min}}{\alpha^r_{\max}\alpha^c_{\max}} < 1. \tag{1.77}$$

GIGA-Win or Learn Fast The most common problems in MARL, regret and convergence, are addressed in gradient-based GIGA-WoLF [119]. GIGA-WoLF is the synergism of GIGA's no-regret property and WoLF-IGA's convergence property [119]. A bound is assigned to test the GIGA-WoLF's regret against the unknown strategy of an opponent agent. For a two-agent, two-action normal-form game, if one agent follows the GIGA-WoLF algorithm and another agent follows the GIGA algorithm, then their strategies does converge to NE. Both the properties are validated theoretically and experimentally in [119]. In GIGA-WoLF, agents must know about the game and should have the model of opponent agent. In almost all the games (except "problematic" Shapley's game), unlike GIGA's strategies, GIGA-WoLF's strategies does converge in self-play to equilibrium.

Dynamic

The dynamic algorithms are categorized as equilibrium dependent and independent. The algorithms based on the equilibrium solution concept are listed below.

Equilibrium Dependent The equilibrium-dependent MARL algorithms are given below.

Nash-Q Learning NQL is the extension of Littman's Minimax-Q learning [100]. In other words, it's the extension of zero-sum-stochastic game to the general-sum-stochastic game. NQL is a MAQL algorithm, which converges under specific conditions. It looks for optimal joint action (NE) in a game. For multiple NEs in the game, the NQL algorithm is fused with other learning techniques to obtain optimal

strategies for the entire team. The adopted framework in [95] is stochastic/Markov games. Markov game is the generalization of the MDP with more than two agents. Unlike, zero-sum game, here in general-sum-stochastic game, an agent's gain is no longer its opponent agent's loss. In general-sum game, an agent's reward depends on other agent's choices and hence, the NE is employed. In NE, an agent cannot deviate unilaterally and it is assumed that there is no communication among the agents. Only agents can observe other agents' strategies and rewards. In addition, the state-transition probabilities and reward functions are unknown. The NQL algorithm is designed in such a way that all the agents converge to the NE with restrictions. NQL is guaranteed that all the agents converge to the NE. But for multiple NE solutions, it is not guaranteed that all the agents converge the same NE. In [80], Filar and Vrieze proposed that every general-sum discounted stochastic game possess at least one equilibrium point in stationary strategy. Unlike single agent Q-learning [84] and Minimax Q-learning [100], in NQL the Q-learning update rule for agent i is given in (1.78).

$$Q_i(S,A) \leftarrow (1-\alpha)Q_i(S,A) + \alpha\left[r_i(S,A) + \gamma\, NashQ_i\left(S'\right)\right], \quad \forall i, \tag{1.78}$$

where

$$NashQ_i\left(S'\right) = \pi_1\left(S'\right)\cdots\pi_m\left(S'\right)\cdot Q_i\left(S'\right). \tag{1.79}$$

An online version of NQL and simulation results on Grid game 1 and 2 are given in [71]. The NQL for general-sum-stochastic game is given in Algorithm 1.17. The convergence proof of Algorithm 1.17 is given [71].

Algorithm 1.17 NQL in General-sum Game

Input: Action $a_i \in A_i$ at $s_i \in S_i$, $\forall\, i$, learning rate $\alpha \in [0, 1)$ and discount factor $\gamma \in [0, 1)$;
Output: Optimal Q-value $Q_i^*(S, A)$, $\forall i$; //$S = \{s_i\}_{i=1}^m$, $A = \{a_i\}_{i=1}^m$;
Initialize: $Q_i(A, A) \leftarrow 0$, $\forall\, i$;
Begin
  Repeat
    Choose an action $a_i \in A_i$, $\forall\, i$;
    Receive immediate reward $r_i(S, A)$, $\forall\, i$;
    Update: $Q_i(S, A) \leftarrow (1-\alpha)\, Q_i(S, A) + \alpha[r_i(S, A) + \gamma\, NashQ_i(S')]$, $\forall\, i$ and $S \leftarrow S'$;
    $Q_i^*(S, A) \leftarrow Q_i(S, A)$, $\forall i$; //$NashQ_i(S') = \pi_1(S') \ldots \pi_m(S') \ldots Q_i(S')$
  Until $Q_i(S, A)$, $\forall\, i$ converge;
End.

Algorithm 1.18 Correlated-Q Learning

Input: Action $a_i \in A_i$ at state $s_i \in S_i$ for all the agents learning rate $\alpha \in [0, 1)$ and discount factor $\gamma \in [0, 1)$;
Output: Optimal Q-value $Q_i^*(S, A), \forall i$;
Initialize: $Q_i(S, A) \leftarrow 0, \forall i$;
Begin
  Repeat
    Choose an action $a_i \in A, \forall i$;
    Receive immediate reward $r_i(S, A), \forall i$;
    Update: $Q_i(S, A) \leftarrow (1-\alpha) Q_i(S, A) + \alpha[r_i(S, A) + \gamma V_i(S')], \forall i$ and $S \leftarrow S'$;
      $V_i(S') = CE(Q_1(S'), Q_2(S'), \ldots, Q_m(S')), \forall i$;
    $Q_i^*(S, A) \leftarrow Q_i(S, A), \forall i$;
    Until $Q_i(S, A), \forall i$; converge;
End.

Correlated-Q Learning In [72], Greenwald and Hall introduced a MAQL algorithm, namely Correlated-Q Learning (CQL). In CQL, Q-value of an agent updates at CE. CQL generalizes both NQL and FFQ in general-sum-stochastic games. If NE and CE do not intersect, then the agent receives less reward at NE compared to the same at CE. Four variants of CE are defined in [72] and the definition of CE is given in Definition 1.18. The algorithm for CQL is given in Algorithm 1.18. Convergence analysis of the above equilibria in the framework of Markov games are done in [72].

Asymmetric-Q Learning In [120], Ville proposed Asymmetric-Q Learning (AQL) algorithm, where an agent leads the follower agents by providing the information about the follower agents' strategy to the follower agents. AQL offers the following benefits:

- In each state the leader has unique equilibrium point.
- Asymmetric Q-learner always achieves the PSNE very fast. Though MSNE exists.
- The AQL algorithm enjoys the lower space and computational requirements than conventional algorithms.

In [120], the existing MAQL algorithms are divided into three clusters. One is the methods utilizing the direct gradients of agents' value function. Second one is the methods that estimate the value functions and then use this estimate to compute

equilibrium of the process. Last one is the use of direct policy gradients. The AQL algorithm is developed by Stackelberg equilibrium (SE) [44]. The algorithm for the leader and the follower are given in Algorithms 1.19 and 1.20. The leader agents are capable to maintain all the agents' Q-tables. However, the follower agents are

Algorithm 1.19 Asymmetric-Q Learning for the Leader

Input: Action $a_i \in A_i$ at state $s_i \in S_i$ for all the agents learning rate $\alpha \in [0, 1)$ and discount factor $\gamma \in [0, 1)$;
Output: Optimal Q-value $Q_i^*(S, A), \forall i$;
Initialize: $Q_i(S, A) \leftarrow 0, \forall i$;
Begin
  Repeat
    Choose an action $a_i \in A, \forall i$;
    Receive immediate reward $r_i(S, A), \forall i$;
    Update: $Q_i(S, A) \leftarrow (1-\alpha) Q_i(S, A) + \alpha[r_i(S, A) + \gamma V_i(S')], \forall i$ and $S \leftarrow S'$;
      $V_i(S') = SE(Q_1(S'), Q_2(S'), \ldots, Q_m(S')), \forall i$;
  $Q_i^*(S, A) \leftarrow Q_i(S, A), \forall i$;
  Until $Q_i(S, A), \forall i$; converge;
End.

Algorithm 1.20 Asymmetric-Q Learning for the Follower

Input: Action $a_i \in A_i$ at state $s_i \in S_i$ for all the agents learning rate $\alpha \in [0, 1)$ and discount factor $\gamma \in [0, 1)$;
Output: Optimal Q-value $Q_i^*(S, A), \forall i$;
Initialize: $Q_i(S, A) \leftarrow 0, \forall i$;
Begin
  Repeat
    Choose an action $a_i \in A, \forall i$;
    Receive immediate reward $r_i(S, A), \forall i$;
    Update: $Q_i(S, A) \leftarrow (1-\alpha) Q_i(S, A) + \alpha\left[r_i(S, A) + \gamma \max_{A'} Q_i(S', A')\right]$ and $S \leftarrow S'$;
  $Q_i^*(S, A) \leftarrow Q_i(S, A), \forall i$;
  Until $Q_i(S, A), \forall i$; converge;
End.

not able to maintain all the agents' Q-values and hence, they just maximize their reward. Experiments are performed in the grid world environment to demonstrate the superiority of the AQL algorithm.

Friend-or-Foe Q-learning In [121], Littman proposed one variant of MAQL algorithm, namely FFQ algorithm with a strong convergence guarantee compared with NE in the framework of general-sum-stochastic game, where agents are instructed to consider other agents' either as a friend or foe. Though, FFQ learning is an improvement over the Nash-Q. In FFQ, two variants' of NE are employed. One is adversarial equilibrium and another is coordination equilibrium. In Minimax-Q (zero-sum game) [100], all the equilibria are adversarial equilibrium. However, in general-sum game, all the equilibria are not coordination equilibrium. Coordination equilibrium provides the highest possible reward of agent i, $i \in [1, m]$ given in (1.80) [121].

$$R_i(\pi_1, ..., \pi_m) = \max_{a_1 \in A_1, ..., a_m \in A_m} R_i(a_1, ..., a_m). \tag{1.80}$$

Except fully cooperative game, coordination equilibrium need not always exist. The adversarial and coordination equilibria are explained in Figure 1.25. The difference between the Nash operation and the maximization or minimax operations is that the latter two have unique solutions. However, the Nash operation offers two variant of solutions: adversarial and coordination equilibrium depending on the problem type.

Two Propositions are proposed and proved in [121]. As per the Propositions, if a one-stage game has a coordination/adversarial equilibrium, then all of the coordination/adversarial equilibrium have same value. There exist two conditions for convergence [121]. In summary, the conditions statement is that for a game there exists either adversarial/coordination equilibrium. Later, two stronger conditions of convergence are proposed in [95, 121]. These conditions can be summarized as follows. There exists an adversarial/coordination equilibrium in a game and every game is defined by the Q-functions adapted during the learning phase. The later conditions are also not sufficient to guarantee convergence. Hu and Wellman [95] state two theorems that by the latter two conditions Nash-Q converges to Nash-Q equilibrium until all the equilibria are adapted during the learning phase are unique. Also by the latter two conditions, Nash-Q converges to NE, until the required equilibria are employed in (1.82).

Now, in FFQ algorithm, $Nash\ Q_i(S^{/})$ are given in (1.81) and (1.82) for Friend-Q (coordination equilibrium) and Foe-Q (adversarial equilibrium), respectively.

$$Nash\ Q_i\left(S^{/}\right) = \max_{A^{/}} \sum_{A^{/}} P\left(A^{/}\right) \cdot Q_i\left(S^{/}, A^{/}\right), \tag{1.81}$$

Algorithm 1.21 Friend-or Foe-Q Learning

Input: Action $a_i \in A_i$ at state $s_i \in S_i$ for all the agents learning rate $\alpha \in [0, 1)$ and discount factor $\gamma \in [0, 1)$;
Output: Optimal Q-value $Q_i^*(S, A), \forall i$;
Initialize: $Q_i(S, A) \leftarrow 0, \forall i$;
Begin
  Repeat
    Choose an action $a_i \in A, \forall i$;
    Receive immediate reward $r_i(S, A), \forall i$;
    Evaluate $Nash\ Q_i(S')$, $\forall i$ by (1.81) and (1.82) respectively for Friend-Q and Foe-Q
    Update: $Q_i(S, A) \leftarrow (1-\alpha) Q_i(S, A) + \alpha[r_i(S, A) + \gamma Nash\ Q_i(S')]$, $\forall i$ and $S \leftarrow S'$;
  $Q_i^*(S, A) \leftarrow Q_i(S, A), \forall i$;
  Until $Q_i(S, A), \forall i$; converge;
End.

$$Nash\ Q_i\left(S^{/}\right) = \max_{a_1, \ldots, a_x} \min_{a_1, \ldots, a_y} \sum\nolimits_{\forall A^{//}} P\left(A^{//}\right) Q_i\left(S^{/}, A^{//}\right), \tag{1.82}$$

where $A^{/} = < a_1, ..., a_m >$, $A^{//} = < a_1, ..., a_x, a_1, ..., a_y >$, and y refers to the number of foes (opponent agents). The convergence of FFQ learning is subject to that the Nash operator is max or minimax operator [121]. Like NQL, for simulation purpose two grid games are employed [95, 121] in FFQ. Six different variants' of opponents are described in [121]. Though, Nash-Q and FFQ cannot fix the problem of finding equilibria, if neither coordination nor adversarial equilibrium exists. The algorithm for FFQ learning is given in Algorithm 1.21.

Negotiation-Based Q-learning In [122], Hu et al. proposed a MARL without mutually sharing their value functions. Authors in [122] mentioned that mutual exchange of value function is impractical because of the local restriction of the system and privacy of the agents in case of distributed agents. Doing so appears impossible to evaluate equilibrium in a one short game. In the above circumstances, authors propose a multi-step negotiation process to evaluate three types of pure strategies: PSNE, equilibrium-dominating strategy profile (EDNP), and nonstrict EDNP, instead of computing the computationally expensive MSNE. It is also shown that abovementioned three strategies are symmetric Meta strategies. Fusing the above techniques, Hu et al. proposed NegoQ in [122].

NegoQ deals with pure strategy equilibrium. However, in some games (e.g. rock-paper-scissor game, Example 1.3), PSNE does not exist. Another hindrance is that a strategy may be Pareto dominated and so not a PSNE. In Prisoners' Dilemma, only one PSNE (C, C) exist as shown in Figure 1.34. Though (D, D) is the better choice, but (D, D) is the Pareto optimal and not a PSNE. In this regard, a strategy profile Pareto dominates NE, i.e. EDNP is defined in Definition 1.19.

Definition 1.19 In an $m-$agent ($m \geq 2$) normal-form game, a joint action $A \in \{A\}$ is an EDNP if there is a PSNE $A_N \in \{A\}$ such that

$$Q_i(A) \geq Q_i(A_N), \quad i = [1, m]. \tag{1.83}$$

By Definition 1.19 one can conclude that each agent following EDNP receives more reward than the same by following PNSE.

Before defining the nonstrict EDSP, a normal-form game with the same is given in Figure 1.45. In Figure 1.45, there are two PSNEs: (a_1, b_1) and (a_2, b_2). It is apparent that the strategy profile (a_1, b_3) and (a_3, b_3) provide a greater reward to A than (a_1, b_1) and a greater reward to B than (a_2, b_2), respectively. So, the priority of (a_1, b_3) and (a_3, b_3) are more than (a_1, b_1) for A and (a_2, b_2) for B, respectively. Hence, for A and B the nonequilibrium strategy profile (a_1, b_3) and (a_3, b_3) partially dominate the existing PSNE. In [122], Hu et al. defined them as nonstrict EDSP as given in Definition 1.20.

Definition 1.20 In an $m-$agent ($m \geq 2$) normal-form game, a joint action $A \in \{A\}$ is an EDNP if there is a PSNE $A_N^i \in \{A\}$ such that

$$Q_i(A) \geq Q_i\left(A_N^i\right), \quad i = [1, m]. \tag{1.84}$$

In the multistep negotiation process of computing the abovementioned three pure strategy profiles, agents exchange their preferences of joint actions among themselves in terms of binary answers. An illustration of the multistep negotiation process is given in Figure 1.46. In Figure 1.46, "Y" and "N" represent as yes and no, respectively. A joint action is pure strategy profile if and only if both the agents' responses are yes. The negotiation process comprises of three types: (i) negotiation for finding the set of PSNE, (ii) negotiation for finding the set of

Figure 1.45 Nonstrict EDNP in normal-form game.

		B → b_1	b_2	b_3
A ↓	a_1	**(20,40)**	(4,22)	(29,30)
	a_2	(18,9)	**(36,19)**	(7,4)
	a_3	(17,26)	(15,38)	(27,38)

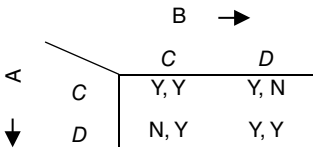


Figure 1.46 Multistep negotiation process between agent A and B.

nonstrict EDSP, and (iii) negotiation for choosing equilibrium (joint action) from the sets obtained by the above two steps. Evaluation of EDSP follows from the evaluation of the nonstrict EDSP, as EDSP is a special case of nonstrict EDSP. The negotiation to evaluate the PSNE for agent i is given in Algorithm 1.22. The negotiation to evaluate the nonstrict EDSP for agent i is given in Algorithm 1.23. Based on the Negotiation algorithms (Algorithms 1.22 and 1.23) to evaluate the pure strategy profiles the NegoQ algorithm for a Markov game is given in Algorithm 1.24. The superiority of

Algorithm 1.22 Negotiation to Evaluate the PSNE for Agent *i* in a Normal-form Game

Input: Action $a_i \in A_i$ only for the agent $i \in [1, m]$ and $Q_i(A)$;
//$A \in \{A\} = \times_{i=1}^{m} A_i$
Output: PSNE set $\{A_N\}$;
Initialize: $\{A_N\} \leftarrow \varphi$;
Evaluate maximal reward set for agent iMS_i;
For all $A_{-i} \in \{A_{-i}\}$

$$a_i = \arg\max_{a_i'} Q\left(a_i', A_{-i}\right);$$

$$\{A_N\} \leftarrow \{A_N\} \cup \{a_i, A_{-i}\};$$

End For
For all joint action $A \in \{A_N\}$
Ask remaining agents that is $\{A_N\}$ includes A;a
If $\{A_N\}$ does not include A **then**
$\{A_N\} \leftarrow \{A_N\} \setminus \{A\}$;
Inform other agents to exclude A from their $\{A_N\}$ sets
End If
End For
For all joint action A' received from remaining agents
If A' belongs to MS_i **then**
Response as yes to the remaining agents;
else
Response as no to the remaining agents;
End If
End For

Algorithm 1.23 Negotiation to Evaluate the Nonstrict EDSP for Agent *i* in a Normal-form Game

Input: Action $a_i \in A_i$ only for the agent $i \in [1, m]$, $\{A_N\}$ from Algorithm 1.13 and $Q_i(A)$; //$A \in \{A\} = \times_{i=1}^{m} A_i$
Output: nonstrict EDSP set $\{A_{nP}\}$;
Initialize: $\{A_{nP}\} \leftarrow \varphi$;
$\{X\} \leftarrow A \backslash \{A_N\}$;
For each PSNE $A_N \in \{A_N\}$
    For each joint action $A \in \{X\}$;
        If $Q_i(A) \geq Q_i(A_N)$ **then**
            $\{X\} \leftarrow \{X\} \backslash \{A\}$;
            $\{A_{nP}\} \leftarrow \{A_{nP}\} \cup \{A\}$;
        End If
    End For
End For
For all joint action $A \in \{A_{nP}\}$
    Ask remaining agents that is A_{nP} includes A;
    If answer is no **then**
        $\{A_{nP}\} \leftarrow \{A_{nP}\} \backslash \{A\}$;
    End If
End For
For all joint action A' received from remaining agents
    If A' belongs to A_{nP} **then**
        Response as yes to the remaining agents;
    else
        Response as no to the remaining agents;
    End If
End For

Algorithm 1.24 Negotiation-Q Learning for Agent *i* in a Markov Game

Input: Joint **action** space $\{A\}$, number of agents' m, state space $\{S\}$, learning rate α, discounting factor γ and exploration rate ε;
Output: Optimal joint Q-value $Q_i^*(S, A)$;
  Initialize: $Q_i(S, A) \leftarrow 0$;
Begin
  Repeat
    Negotiate with remaining agents employing Algorithms 1.13 and 1.14;
    Select the pure strategy equilibrium A' using ε-greedy;
    Receive experience tuple $<S, A, r_i(S, A), S'>$; // $r_i(S, A)$ and S' are the immediate reward and next joint state
    Update: $Q_i(S, A) \leftarrow (1-\alpha) Q_i(S, A) + \alpha [r_i(S, A) + \gamma Q_i(S', A')]$, and $S \leftarrow S'$;
    $Q_i^*(S, A) \leftarrow Q_i(S, A)$;
  Until $Q_i(S, A)$; converges;
End.

Algorithm 1.24 is tested in grid-world maps over the state-of-the-art reference algorithms.

MAQL with Equilibrium Transfer Hu et al. [123] identified that agents evaluate the same equilibrium (NE or CE) at a joint state for different one-shot games. Here, two equilibria are declared as same if and only if the Euclidian distance between the probability distribution of the strategies is less than a predefined threshold. Reuse of the previously computed equilibrium (or equilibrium transfer) decreases as the convergence time of the equilibrium-based MAQL decreases with negligible transfer loss. Suppose G and $G^{/}$ are two one-short games that visit the same joint state S. Now, the Euclidian distance between the equilibrium strategy p of G and $p^{/}$ of $G^{/}$ are given in (1.85) and (1.86), respectively, for NE and CE.

$$d^{NE}\left(p, p^{/}\right) = \sqrt{\sum_{i=1}^{n} \sum_{a_i \in A_i} \left(p_i(a_i) - p_i^{/}(a_i)\right)^2}, \tag{1.85}$$

$$d^{CE}\left(p, p^{/}\right) = \sqrt{\sum_{A \in \{A\}} \left(q(A) - q^{/}(A)\right)^2}. \tag{1.86}$$

If the $d^{NE}(p, p^{/})$ or $d^{CE}(p, p^{/})$ is smaller than a threshold, then p and $p^{/}$ are considered as identical in G and $G^{/}$. Hence, by equilibrium transfer one can directly use p in $G^{/}$. As computation of equilibrium is more expensive than checking, hence, there is a significant saving in computational cost. Hu et al. [123] measures the equilibrium transfer loss and based on that loss the equilibrium transfer condition is defined. Let p^{*} and q^{*} denote the NE and CE of G and $G^{/}$, respectively. Now, loss because of transferring the equilibrium p^{*} and q^{*} from G to $G^{/}$ is given by (1.87) and (1.88), respectively.

Algorithm 1.25 Equilibrium Transfer-Based MAQL

Input: Action $a_i \in A_i$ at state $s_i \in S_i$ for all the agents learning rate $\alpha \in [0, 1)$, discount factor $\gamma \in [0, 1)$, exploration factor ε, threshold of transfer loss τ, G_c be the one-short game at joint state S and p^{*} previously computed equilibrium at S;
Output: Optimal joint Q-value $Q_i(S, A), \forall i$; //$S \in \{S\} = \times_{i=1}^{m} S_i$ and $A \in \{A\} = \times_{i=1}^{m} A_i$
  Initialize: $Q_i(S, A) \leftarrow 0, \forall i$;
Repeat
  If joint state S has been visited
    then evaluate maximum utility loss ε^{Ω}, $\Omega \in \{NE, CE\}$ for transferring to G_c;
  Else

$\varepsilon^{\Omega} \leftarrow +\infty$;
End if
If $\varepsilon^{\Omega} > \tau$
Then evaluate p^* for G_c;
Else
Reuse p^* in G_c;
End if
Select joint action, A sampled from p^*;
Receive experience (S, A, r_i, S'), $\forall i$;
Evaluate equilibrium p' for the next joint state S';
Evaluate $V_i(S') \leftarrow$ expected value of p' in S', $Q_i(S, A) \leftarrow (1-\alpha) Q_i(S, A) + \alpha(r_i + \gamma V_i(S'))$ and $S \leftarrow S'$;
Until $Q_i(S, A)$, $\forall i$ converge;

$$\varepsilon^{NE} = \max_{i \in N} \max_{a_i \in A_i} \left(Q_i^{G'}(a_i, p^*_{-i}) - Q_i^{G'}(p^*) \right), \tag{1.87}$$

$$\varepsilon^{CE} = \max_{i \in N} \max_{a_i \in A_i} \max_{a_i' \in A_i} \sum_{A_{-i}} q^*(a_i, A_{-i}) \times \left[Q_i^{G'}\left(a_i, A_{-i}\right) - Q_i^{G'}\left(a_i', A_{-i}\right) \right]. \tag{1.87}$$

Here, $Q_i^{G'}$ refers to the Q-value of agent i in G'.
Now the transfer loss condition for NE, p^* for an agent i is given by

$$\begin{aligned} Q_i^{G'}(p^*) + \varepsilon^{NE} &\geq Q_i^{G'}(p^*) + \max_{a_i \in A_i} \left(Q_i^{G'}(a_i, p^*_{-i}) - Q_i^{G'}(p^*) \right) \\ &= Q_i^{G'}(p^*) + \max_{a_i \in A_i} Q_i^{G'}(a_i, p^*_{-i}) - Q_i^{G'}(p^*) \\ &= \max_{a_i \in A_i} Q_i^{G'}(a_i, p^*_{-i}). \end{aligned} \tag{1.89}$$

Similarly, for CE, the following condition can be derived:

$$\sum_{A_{-i}} q^*(a_i, A_{-i}) \times Q_i^{G'}(a_i, A_{-i}) + \varepsilon^{CE} \geq \sum_{A_{-i}} q^*(a_i, A_{-i}) \times Q_i^{G'}\left(a_i', A_{-i}\right). \tag{1.90}$$

The algorithm for equilibrium transfer-based MAQL is given in Algorithm 1.25. Superiority of Algorithm 1.25 is tested in Grid World game, Wall game, and Soccer game.

Equilibrium Independent Equilibrium-independent MARL algorithms are again categorized based on the learning rate selection given below.

Variable Learning Rate RL algorithms with variable learning rate are given below.

Win or Learn Fast Policy Hill-Climbing In [124], Bowling and Veloso proposed WoLF policy hill-climbing algorithm for stochastic game in the presence of other adaptive agents, satisfying rationality and convergence. *Rationality* indicates that all agents' policies converge to stationary policies and then the learning algorithm will converge to a stationary policy, which is best response to their policies [124]. The convergence property states that agents necessarily converge to a stationary policy. Also, if all agents are rational and convergent, then it is guaranteed to converge NE. The learning algorithms in [31] and [125] either converge to a suboptimal policy or does not converge. Proposed WoLF is based on the principle of *learn quickly while losing and learn slowly while wining*.

Policy hill-climbing (PHC) is a straightforward extension of Q-learning to handle mixed strategies. The PHC algorithm is given in Algorithm 1.26. PHC learns the most recent mixed strategy. The updating of the mixed strategy in PHC is done by selecting the highest valued action as per the learning rate $\delta \in (0, 1]$. For $\delta = 1$, the algorithm behaves as single agent Q-learning. Both Q-values and the strategy are convergent following single agent Q-learning.

The main contribution of the proposed algorithm in [124] is the extension of PHC algorithm by employing a variable learning rate and the WoLF principle.

Algorithm 1.26 Policy Hill-Climbing (PHC)

Input: Action $a_i \in A_i$ at state $s_i \in S_i$ for all the agents learning rate $\alpha \in [0, 1)$ and discount factor $\gamma \in [0, 1)$;
Output: Optimal policy $\pi_i^*(S, A)$;
Initialize: $Q_i(S, A) \leftarrow 0$ and $\pi_i(S, A) \leftarrow \frac{1}{|A_i|}$;
Begin
  Repeat
    Choose an action $a_i \in A$ with probability $\pi_i(S, A)$;
    Receive immediate reward $r_i(S, A)$;
    Update: $Q_i(S, A) \leftarrow (1-\alpha)Q_i(S, A) + \alpha\left[r_i(S, A) + \gamma \max_{A'} Q_i(S', A')\right]$,
    $S \leftarrow S'$ and

$$\pi_i(S, A) \leftarrow \pi_i(S, A) + \begin{cases} \delta, & If\ A = \arg\max_A Q(S, A') \\ \frac{-\delta}{|A_i| - 1}, & \text{otherwise} \end{cases}$$

    $\pi_i^*(S, A) \leftarrow \pi_i(S, A)$;
  Until $\pi_i^*(S, A)$ converges;
End.

In variable learning rate, the learning rate is used by the learning algorithm and is tuned in such a way so that the rationality is maintained. The WoLF principle motivates to learn quickly while losing and slowly while winning [124]. The WoLF-PHC algorithm employs two learning rate: losing learning rate δ_l and winning learning rate δ_w, where $\delta_l > \delta_w$. The winning/losing situation of the agent is determined by contrasting the current reward and the average reward taken over the time. If the agent is losing, then larger learning rate δ_l is employed. The WoLF-PHC [124] algorithm is given in Algorithm 1.27. The convergence and rationality of the WoLF-PHC algorithm is tested in Matrix games, Grid world game, and Soccer game. In all frameworks, WoLF-PHC outperforms reference algorithms.

Policy Dynamic-Based Win or Learn Fast (PD-WoLF) IGA [111] learner converges to NE rationally but they are not convergent to NE for all the general-sum games. Later, IGA was extended to WoLF-IGA in [107] and its convergence proof is shown in [107] for a 2×2 game assuming agents know the equilibrium policies of other agents. In [126], Banerjee and Peng did experimental-based comparisons of the

Algorithm 1.27 Win or Learn Fast-PHC (WoLF-PHC)

Input: Action $a_i \in A_i$ at state $s_i \in S_i$ for all the agents learning rate α, $\delta_l > \delta_w$ and discount factor $\gamma \in [0, 1)$;
Output: Optimal policy $\pi_i^*(S, A)$;
Initialize: $C(S) \leftarrow 0$, $Q_i(S, A) \leftarrow 0$ and $\pi_i(S, A) \leftarrow \frac{1}{|A_i|}$;
Begin
  Repeat
    Choose an action $a_i \in A$ with probability $\pi_i(S, A)$;
    Receive immediate reward $r_i(S, A)$;
    Update: average policy $\overline{\pi}$, $C(S) \leftarrow C(S) + 1$, $S \leftarrow S'$,
    $\overline{\pi}(S, A') \leftarrow \overline{\pi}(S, A') + \frac{1}{C(S)}\left[\pi_i(S, A') - \overline{\pi}(S, A')\right]$

$$\text{and} \quad \pi_i(S, A) \leftarrow \pi_i(S, A) + \begin{cases} \delta, & \text{If } A = \underset{A}{\operatorname{argmax}} Q(S, A'); \\ \dfrac{-\delta}{|A_i| - 1}, & \text{otherwise;} \end{cases}$$

$$\delta = \begin{cases} \delta_w, & \text{If } \sum_A \pi_i(S, A) Q_i(S, A) > \sum_A \overline{\pi}(S, A) Q_i(S, A) \\ \delta_l, & \text{otherwise} \end{cases};$$

    $\pi_i^*(S, A) \leftarrow \pi_i(S, A)$;
  Until $\pi_i^*(S, A)$ converges;
End.

WoLF and PD-WoLF to establish the superiority of the PD-WoLF both in the bimatrix and the general-sum games.

From Section "Infinitesimal Gradient Ascent," considering the sub case of purely imaginary Eigen values, U and the center (r^*, c^*) are within the unit square. The solution $r(t)$ of (1.72) for unconstraint dynamics [127] is given in (1.91), where the value of B and ϕ depends on the initial values of α, β.

$$r(t) = B\sqrt{u}\cos\left(\sqrt{uu't} + \phi\right) + r^*. \tag{1.91}$$

PD-WoLF criteria for a row player (agent) are given by (1.92).

$$\alpha_r(t) = \begin{cases} \alpha_{\min}, & \text{if } \Delta_t \Delta_t^2 < 0 \\ \alpha_{\max}, & \text{otherwise} \end{cases}, \tag{1.92}$$

where $\Delta_t = r_t - r_{t-1}$ and $\Delta_t^2 = \Delta_t - \Delta_{t-1}$. It is apparent that (1.92) is independent of other agents' policies.

Fixed Learning Rate MARL Algorithms with fixed learning rate are given below.

Non-Stationary Converging Policies One major shortcoming of MAQL is the assumption that the environment is stationary. In [128], Michael and Jeffrey proposed the NSCP, where agents are not interested in converging to an equilibrium rather they search for the best-response policy for the non-stationary opponents. NSCP predicts the opponents' non-stationary strategy with precision and act by its best-response strategy with respect to the opponents in the well-known test bench of general-sum-stochastic games (game with multiple joint states) or matrix games (game with one joint state). The MAQL algorithms [71, 72, 81, 95, 100] and [121] either converge to NE or CE. By [129], the equilibrium-based MAQL algorithms are problematic, as the learning stops at the equilibrium point and the equilibrium point is necessarily not a goal point. Also an additional problem arises in the presence of multiple equilibria. The NSCP algorithm aims at adapting an optimal reward considering the presence of other agents. In [130], an agent converges to best-response strategy subject to stationary opponents in two-player general-sum-stochastic games. The NSCP algorithm is given in Algorithm 1.28. Simulation results validate the superior performance of the NSCP with respect to reference algorithms.

Extended Optimal Response Learning The zero-sum-stochastic game proposed by Littman [100] was extended to general-sum-stochastic game by Hu and Wellman [95] and agents converge to NE in stochastic games by these algorithms. On the contrary, in [95] and [100], agents always try to converge to NE ignoring strategies of other agents. Further, all the agents must agree upon to select a NE in the

Algorithm 1.28 Non-Stationary Converging Policies

Input: Action $a_i \in A_i$ at state $s_i \in S_i$, $\forall i$, $i \in [1, m]$, learning rate $\alpha \in [0, 1)$ and discount factor $\gamma \in [0, 1)$;
Output: Optimal Q-value $Q_i^*(S, A)$, $\forall i$;
Initialize: $Q_i(S, A) \leftarrow 0$, $\forall i$ and $\pi^i(S, A) = \frac{1}{|A|}$;
Begin
  Repeat
    Observe the actions taken by all the agents $A \in \{A\}$;
    Receive immediate reward $r_i(S, A)$, $\forall i$;
    Update: other agents' strategy $\pi_{-i}(S, A) = \frac{1}{|A|}$, $\forall i$;
    Select best-response strategy $\pi_i^{br}(S, A)$
    that maximizes $BR(S') = \sum_{a_1}\sum_{a_2}\cdots\sum_{a_m}\pi_i^{br}(S', a_i)\cdot\prod_{\forall -i}\hat{\pi}_i(S', a_i)\cdot Q_i(S', A)$;
    Update: Q-values using the following rules
    $Q_i(S, A) \leftarrow (1-\alpha)Q_i(S, A) + \alpha[r_i(S, A) + \gamma BR(S')]$ and $S \leftarrow S'$;
  Until $Q_i(S, A)$, $\forall i$; converges;
  Obtain $Q_i^*(S, A) \leftarrow Q_i(S, A)$, $\forall i$;
End.

presence of multiple NEs. Thus, the algorithms proposed in [95] and [100] are not adaptable in the above sense. In [131], Nobuo and Akira extended optimal response to EXORL, where agents converge to NE subject to adaptability of other agents. Similar to NQL [95], in EXORL, an agent maintains all agents' Q-tables assuming that it can observe other agents' state–action and reward. EXORL aims at realizing a policy which is optimal response to other agents' policies, where remaining agents are adaptable and attain NE. The EXORL algorithm is given in Algorithm 1.29. JAL [81] learns Q-value due to its own action and estimates teammates' strategy. Let π_i be the strategy of agent i at state S which maximizes (1.93).

$$Q_i(S, \pi_i) = (\pi_i)^T Q_i(S)\overline{\pi}_{-i}(S), \tag{1.93}$$

where $\overline{\pi}_{-i}(S)$ refers to estimate of all agents' joint policy except agent i. Now, if a policy diverges from NE, then the policy may not be suitable to estimate the remaining agents' strategy. This problem is addressed in [131] and the update rule is given by (1.94) and (1.95) tuning the value of ρ.

$$Q_i(S, \pi_i) = (\pi_i)^T Q_i(S)\overline{\pi}_{-i}(S) - \rho\sigma(S, \pi_i), \tag{1.94}$$

Algorithm 1.29 EXORL for Agent *i*

Input: Action $a_i \in A_i$ at $s_i \in S_i, \forall i$, learning rate $\alpha \in [0, 1)$ and discount factor $\gamma \in [0, 1)$;
Output: Optimal Q-value $Q_i^*(S, A), \forall i$; $//S = \{s_i\}_{i=1}^m, A = \{a_i\}_{i=1}^m$;
Initialize: $Q_i(A, A) \leftarrow 0, \pi_i(S, a_i) \leftarrow \frac{1}{|A_i|} \forall i, \overline{\pi}_{-i}(S, A_{-i}) \leftarrow \frac{1}{|\times_{j=1, j\neq i}^{m} A_j|}$;
Begin
Repeat
Choose an action $a_i \in A_i, \forall i$;
Receive immediate reward $r_i(S, A), \forall i$;
Update: $Q_i(S, A) \leftarrow (1-\alpha) Q_i(S, A) + \alpha [r_i(S, A) + \gamma Q_i(S', A)]$, $\forall i$, $S \leftarrow S'$
and $\overline{\pi}_{-i}(S) \leftarrow (1-\beta)\overline{\pi}_{-i}(S) + \beta.\overline{\pi}_{-i}(S')$; $//\overline{\pi}_{-i}(S') = \begin{cases} 1 & \text{if } A_{-i} = A'_{-i} \\ 0 & \text{otherwise} \end{cases}$
Until $Q_i(S, A), \forall i$ converge;
Obtain $Q_i^*(S, A) \leftarrow Q_i(S, A), \forall i$;
End.

where

$$\sigma(S, \pi_i) = \max_{\pi_{-i}} \left[(\pi_i)^T Q_{-i}(S) \pi_{-i} \right] - (\pi_i)^T Q_{-i}(S) \overline{\pi}_{-i}(S), \tag{1.95}$$

here $\sigma(S, \pi_i)$ refers to the possible increase in expected discounted reward of agent i. Hence, to maximize the left part of (1.95), agent i has to maximize the first component of right part and also minimizes the second component of the right part. Also, (1.95) is a piece-wise linear concave function and it has a sole maximal point. It is shown in [131] that by EXORL an agent plays well subject to that the opponent agents play fixed policy considering small value of ρ. The EXORL is verified in Matching Pennies, Presidency Game [80], and Battle of sexes game in [131].

1.3.6 Coordination and Planning by MAQL

In the present book, for multi-robot coordination and planning without any communication among the agents, we focus on the equilibrium-based MAQL as explained in Section "Equilibrium Dependent". Because of the absence of communication among the agents, each agent needs to maintain all the agents' Q-tables at joint state–action space. Figure 1.47 explains the multi-robot coordination and planning mechanism for the well-known stick-carrying problem. Stick-carrying

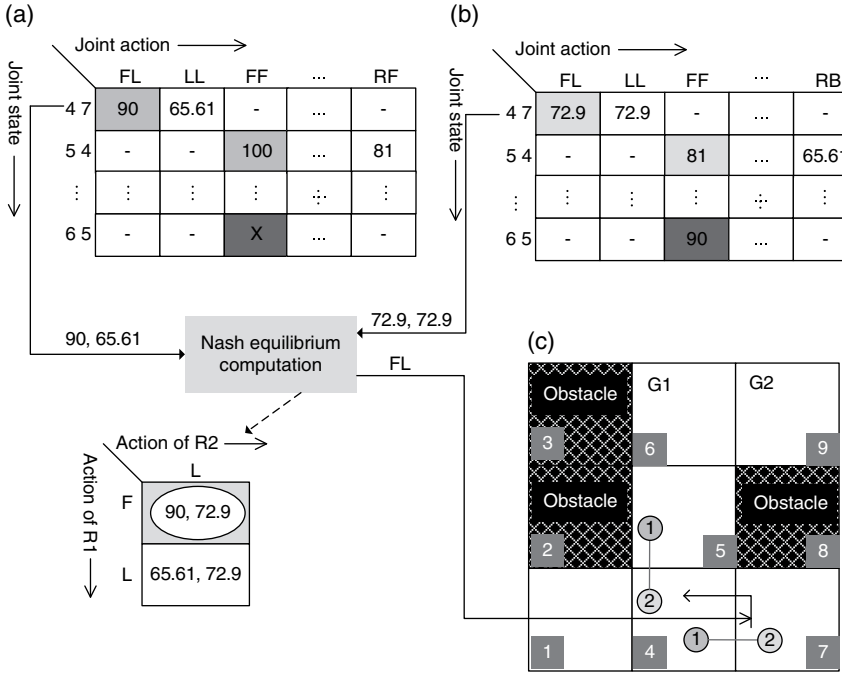


Figure 1.47 Multi-robot coordination for the well-known stick-carrying problem. (a) Joint Q-table of agent 1 by NQL. (b) Joint Q-table of agent 2 by NQL. (c) Stick-carrying by employing (a) and (b).

problem refers to the transportation of a stick from current positions to the desired destination. Presently, twin robots are at a joint state <4, 7> with a stick as shown in Figure 1.47c. As each robot have both robots' Q-tables at joint state–action space, a robot looks for the optimal joint action, i.e. PSNE at <4, 7> by evaluating equilibrium. To evaluate equilibrium, a robot extracts the information from the joint state <4, 7> (Figure 1.47a, b) and PSNE, "FL" is evaluated following the definition of NE as shown in Figure 1.47. Here, both the robots evaluate identical PSNE. Hence, without any communication between the robots, coordination occurred and the stick is shifted to the next joint state <5, 4> because of the joint action "FL" by the robots.

1.3.7 Performance Analysis of MAQL and MAQL-Based Coordination

The MAQL algorithms illustrated above have addressed several challenges of the MAQL. The main challenges of MAQL are suitable action selection for balancing exploration/exploitation, update policy selection for adaptation of the Q-table in

joint state–action space, equilibrium selection among multiple equilibria, and the exponential increase in the space and time complexity, with the increase in number of agents. In this regard, to measure the performance of a MAQL over contender MAQL algorithms, the following metrics are summarized for the abovementioned MAQL.

In JAL [81], the Boltzmann strategy is extended to the OB, WOB, and their combination. The superiority of the JAL with the combined method is tested considering the average accumulated reward as the performance metric. The superiority of the FMQ heuristic is measured considering convergence to the optimal joint action as the performance metric. In Team-Q learning [92], the average reward of agents is maximized over the learning epoch. The Distributed Q-learner [91] converges to the optimal joint action with less storage and computational cost. Therefore, in Distributed Q-learning, computational cost and storage requirement are the performance metrics. In OAL [96] algorithm, agents select the optimal NE among multiple NE with probability one. Hence, in OAL, optimal equilibrium selection is the metric. In SCQL [97], the Q-tables are sparsely maintained and performance of the SCQL is measured over reference algorithms in terms of the computational cost and storage requirement. In SQL [98], the metric is the steps required to reach the goal state from the starting state, i.e. selection of the right joint action without any behavior conflict among the agents. In FMRQ [99], agents achieve the coordination-type optimal NE to maximize the system performance in terms of average steps per episode for box-pushing problem and average rewards per episode for distributed sensor network problem. In Minimax-Q learning algorithm [100], both the agents learn optimal policies and efficiency of the algorithm is tested in the framework of a two-player grid game by measuring the winning percentage of the game by the agent in an episode. Performance of the HAMRL algorithm [101] is measured in terms of the convergence speed. FP [105] addressed the equilibrium selection problem in coordination game. The performance of the Meta strategy [106] is measured in terms of the average reward achieved by the agents. AWESOME [108] learns the best response (NE) considering a stationary opponent and its performance is measured against FP in terms of the distance to equilibrium and distance to the best response. In Hyper-Q learning [110], online Bellman error and average reward variation with respect to the learning epoch are considered as the performance metrics. In [111], IGA proposed a scheme by which agents conditionally converge to the NE. Performance of the GIGA [118], WoLF-IGA [107], and GIGA-WoLF [119] algorithms are measured in terms of the convergence rate. In NQL [95], percentage of NE achieved in a game is considered as the performance metric. In CQL [72], mean Q-value difference is the performance metric. In AQL [120], change in Q-values of the agents with the learning epoch is considered as the performance metric. The FFQ [121] always converges to a NE and converging to a NE is a metric. Average reward with the episode and number

of learning epoch required per episode are the metrics in NegoQ. In the equilibrium transfer-based MAQL [123], three metrics are considered. First one is the learning speed, second one is the improved average reward, and finally the last one is the reduction in the space complexity. In WoLF-PHC [124], the policy either converges to NE or to a suboptimal NE and percentage of winning a game by an agent is considered as the performance metric. In PD-WoLF [111], average reward is the performance metric during the learning phase. Average time required to complete a task is considered the performance metric during learning in case of NSCP [128]. In EXORL [100], policy and Q-value learned with the learning epoch are considered as the performance metric.

In MAQL-based coordination, agents re-evaluate the NE/CE as explained in Section 1.3.6. As the computational cost of evaluating the NE/CE is very high, run-time complexity is one performance metric in the MAQL-based coordination. On the other hand, space-complexity, successful completion of the task, system resource utilization, and the like are considered as the performance metrics during the MAQL-based coordination [98].

1.4 Coordination by Optimization Algorithm

One common bottleneck of the search-based coordination and MARL-based algorithms is the memory requirement and suboptimal solution. Such bottlenecks are addressed by the Swarm Intelligence (SI) [61, 62] and EA [62]. The advantages of the SI algorithms are Scalability, Adaptability, Collective Robustness, and Individual Simplicity. The scalability of the SI algorithms are remarkable, as the control mechanism adopted by the SI algorithms does not depend upon the swarm size, until the swarm size is not too small [45]. The SI algorithm has very fast response to the rapidly changing environment by employing the auto-configuration and self-organization capabilities, which allow the swarms to adapt online with the dynamic environment [66]. Collective robustness indicates that the SI algorithms are distributed and hence, there is no possibility of single point failure [67]. In spite of very simple behavior of every swarm in any SI algorithm, the group of a swarm can achieve sophisticated group behavior [67]. Particle Swarm Optimization (PSO) algorithm and Firefly algorithm (FA) are two examples of SI algorithms. In PSO, the fitness function is not differentiable and is employed to obtain quality solution for high-dimensional problems faster than other alternatives. However, there is a high probability to be trapped in local optima in high-dimensional problems. On the other hand, the FA has a very high probability of exploring the global optima. The advantages of EAs are that they can cope with discontinuities, nonlinear constraints, multi-modalities, and multi-objective optimization problems.

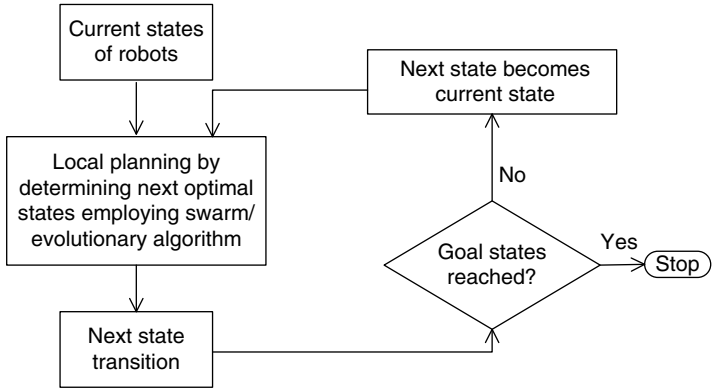


Figure 1.48 Multi-robot local planning by swarm/evolutionary algorithm.

However, the EAs do not provide any guarantee to provide optimal solutions within finite amount of time. Differential evolution is an example of EA. Stability is a very good attribute of DE over the GA. Another is Imperialist Competitive Algorithm (ICA) [67], which is a sociopolitical-based algorithm. ICA has neighborhood movements both in continuous and discrete search-space. However, the solutions provided by the ICA does not guarantee for optimal solution. In addition, the ICA requires tuning more number of parameters as compared with the PSO, FA, and DE. In the above circumstance, hybridization is a good approach. By hybridization, the efficient attributes of two or more algorithms are fused to produce a powerful algorithm. One approach for multi-robot stick-carrying problem is shown in [91], where the hybridization of the motion dynamics of fireflies of the FA [48] into a sociopolitical evolution-based meta-heuristic search algorithm is done and is named as Imperialist Competitive Firefly Algorithm (ICFA). The abovementioned algorithms are implemented for multi-robot coordination following scheme as shown in Figure 1.48. Brief description of the abovementioned algorithms are given below.

1.4.1 PSO Algorithm

In [61], Kennedy and Eberhart proposed a nonlinear function optimization technique following the behavior of flocking birds, namely PSO. Let an n-dimensional nonlinear function given by (1.96) to be optimized. The PSO aims at finding such a $\overline{X}$ so that (1.96) is either maximized or minimized depending upon the problem requirement. So, one can say that the solution of (1.96) is an n-dimensional hyperspace.

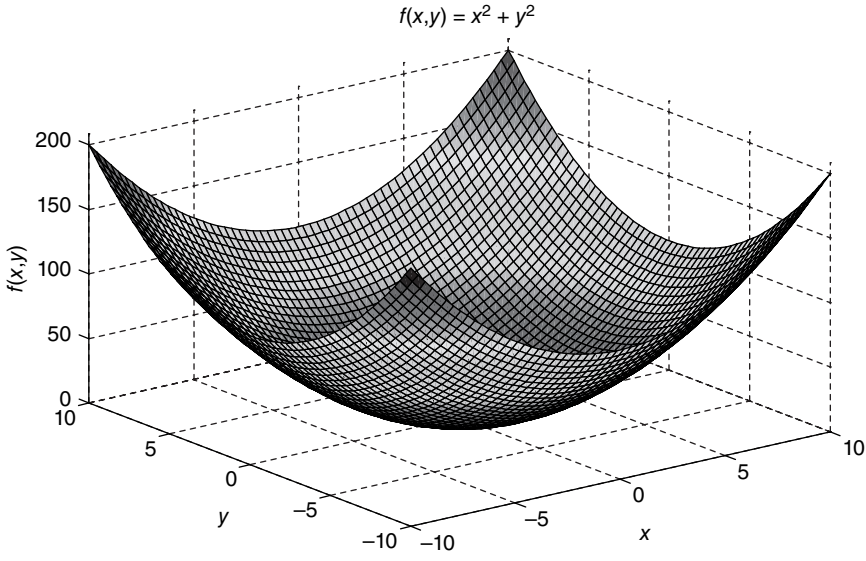


Figure 1.49 Surface plot of (1.97).

$$f\left(\vec{X}\right) = f(x_1, ..., x_n). \tag{1.96}$$

Let us consider a two-dimensional problem as given in (1.97) [48]. In (1.97), $x \in [-10, 10]$ and $y \in [-10, 10]$ and the plot of (1.97) is given in Figure 1.49. It is apparent from Figure 1.49 that (0, 0) is the only solution in the xy plane for which the $f(x, y)$ attains a minimum value of zero. It is quiet easy to identify the minima for the function (1.96) compared to the same for (1.98) [48]. The plot of (1.98) is shown in Figure 1.50. Unlike Figure 1.49, in Figure 1.50, there are multiple optimal points. It is difficult to identify the global optima among them. PSO employs the multi-agent parallel search technique and each agent starts from different initial positions and explores the landscape until a global optima is reached. It is assumed that in PSO, agents can communicate among themselves and share the values of fitness function explored by them.

$$f(x,y) = x^2 + y^2, \tag{1.97}$$

$$f(x,y) = x\sin(4\pi y) + y\sin(4\pi x + \pi) + 1. \tag{1.98}$$

In PSO, each agent flies through the multidimensional landscape with a unique position and velocity at each landscape. The population is initialized with random positions denoted by $\vec{X} = \{x_i\}_{i=1}^{S}$ each having a random velocity $\vec{V} = \{v_i\}_{i=1}^{S}$.

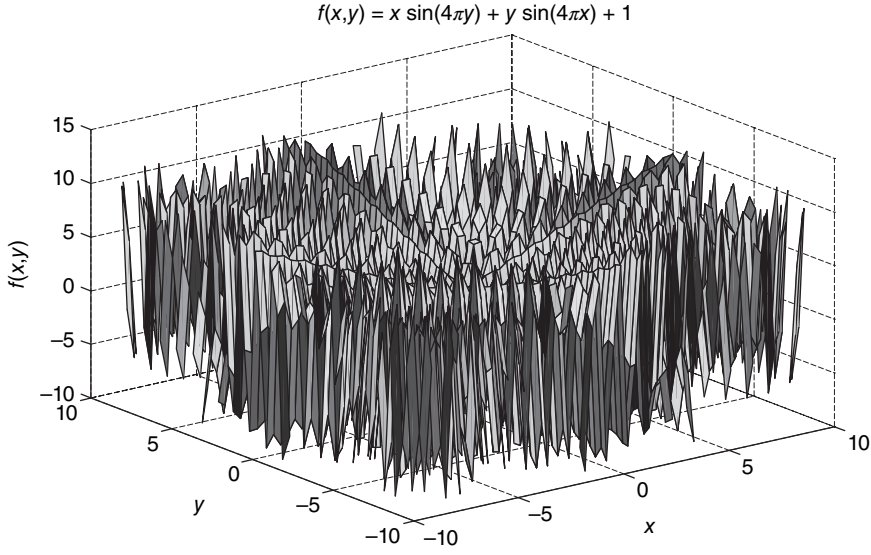


Figure 1.50 Surface plot of (1.98).

The position and velocity of the d-th dimension's i-th particle is given by (1.99) and (1.100), respectively.

$$x_{id}(t+1) = x_{id}(t) + v_{id}(t+1), \tag{1.99}$$

$$v_{id}(t+1) = \omega \cdot v_{id}(t) + C_1 \cdot \varphi_1 \cdot (P_{id}(t) - x_{id}(t)) + C_2 \cdot \varphi_2 \cdot (g_{id}(t) - x_{id}(t)). \tag{1.100}$$

In (1.100), the first component is the initial velocity of the i-th particle. ω refers to the inertial weight factor. C_1 and C_2 are the constant multiplier termed as self-confidence and swarm confidence, respectively. Two random numbers $\varphi_1 \in [0, 1]$ and $\varphi_2 \in [0, 1]$ are introduced in (1.100), which determine the influence of $\vec{p}(t)$ and $\vec{g}(t)$ on (1.100). $\vec{p}(t)$, $\vec{g}(t)$, and $\vec{x}(t)$ are initialized to zero at $t = 0$, i.e. and $\vec{p}(0) = \vec{g}(0) = \vec{x}(0)$. After that the velocity and position of each particle update following (1.99) and (1.100). The algorithm for PSO is given in Algorithm 1.30 [48].

In [69], Pugh et al. proposed the noise-resistance PSO for obstacle avoidance in multi-robot systems. In [70], Pugh modified the noise-resistance PSO [69] by setting

$$x_i^{*'} = x_i^{*''}, \text{if } fitness\left(x_i^{*''}\right) > fitness\left(x_i^{*'}\right), \tag{1.101}$$

Algorithm 1.30 Particle Swarm Optimization (PSO)

Input: Enter the Swarm size (S), values of C_1, C_2, $\varphi_1 \in [0, 1]$, $\varphi_2 \in [0, 1]$, ω and V_{max};
Output: Approximate global optimal position $\vec{X}^*$;
Initialize: Initialized the position and velocity vectors: $\vec{X}_i(0)$ and $\vec{V}_i(0)$;
Begin
While termination condition is not reached **do**
  For i = 1 to S
  Evaluate the fitness $f\left(\vec{X}_i\right)$;
  Update $\vec{p}_i$ and $\vec{g}_i$;
  Adapt position and velocity of the partial by (1.99) and (1.100) respectively.
End For;
End While.
End.

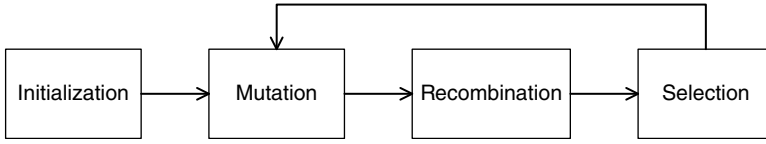


Figure 1.51 Steps of Differential evolution (DE) algorithm [132].

where $x_i^{*'}$ refers to the neighborhood best for particle (here robot) i and $x_i^{*''}$ denotes the new neighborhood best particle.

1.4.2 Firefly Algorithm

In FA [93], a potential solution to an optimization problem is encoded by the position of a firefly in the search space and the light intensity at the position of the firefly corresponds to the fitness of the associated solution. Each firefly changes its position iteratively by flying toward brighter fireflies at more attractive location in the fitness landscape to obtain optimal solutions.

1.4.2.1 Initialization

FA commences with a population P_t of NP, D-dimensional firefly positions, $\vec{X}_i(t) = \{x_{i,1}(t), x_{i,2}(t), x_{i,3}(t), ..., x_{i,D}(t)\}$ for $i = [1, NP]$ by randomly initializing in the search range $\left[\vec{X}^{\min}, \vec{X}^{\max}\right]$ where $\vec{X}^{\min} = \{x_1^{\min}, x_2^{\min}, ..., x_D^{\min}\}$ and $\vec{X}^{\max} = \{x_1^{\max}, x_2^{\max}, ..., x_D^{\max}\}$ at the current generation $t = 0$. Thus, the d-th component of the i-th firefly at $t = 0$ is given by (1.102).

$$x_{i,d}(0) = x_d^{\min} + rand(0,1) \times \left(x_d^{\max} - x_d^{\min}\right), \tag{1.102}$$

where $rand(0, 1)$ is a uniformly distributed random number lying between 0 and 1 and $d = [1, D]$. The objective function value $f\left(\vec{X}_i(0)\right)$ (which is inversely proportional to the light intensity for minimization problem) of the i-th firefly is evaluated for $i = [1, NP]$.

1.4.2.2 Attraction to Brighter Fireflies

Now the firefly $\vec{X}_i(t)$ is attracted toward the positions of the brighter fireflies $\vec{X}_j(t)$ for $i, j = [1, NP]$ but $i \neq j$ such that $f\left(\vec{X}_j(t)\right) < f\left(\vec{X}_i(t)\right)$ for minimization problem.

Now the attractiveness $\beta_{i,j}$ of $\vec{X}_i(t)$ toward $\vec{X}_j(t)$ is proportional to the light intensity seen by adjacent fireflies. However attractiveness $\beta_{i,j}$ decreases exponentially with the distance between the fireflies, denoted by $r_{i,j}$ as given in (1.103).

$$\beta_{i,j} = \beta_0 \exp\left(-\gamma \times r_{i,j}^m\right), \quad m \geq 1, \tag{1.103}$$

where β_0 denotes the maximum attractiveness experienced by the i-th firefly at its own position (i.e. at $r_{i,j} = r_{i,i} = 0$) and γ is the light absorption coefficient, which controls the variation of $\beta_{i,j}$ with $r_{i,j}$. This parameter is responsible for the convergence speed of FA. A setting of $\gamma = 0$ leads to constant attractiveness while γ approaching infinity is equivalent to the complete random search [48]. In (1.103), m is a positive constant representing a nonlinear modulation index. The distance between $\vec{X}_i(t)$ and $\vec{X}_j(t)$ is computed using the Euclidean norm as follows:

$$r_{i,j} = \left\|\vec{X}_i(t) - \vec{X}_j(t)\right\|. \tag{1.104}$$

This step is repeated for $i, j = [1, N]$.

1.4.2.3 Movement of Fireflies

The firefly at position $\vec{X}_i(t)$ moves toward a more attractive position $\vec{X}_j(t)$ occupied by a brighter firefly (i.e. $f\left(\vec{X}_j(t)\right) < f\left(\vec{X}_i(t)\right)$) for $j = [1, N]$ but $i \neq j$ following the dynamic given in (1.105).

$$\vec{X}_i(t+1) = \vec{X}_i(t) + \beta_{i,j} \times \left(\vec{X}_j(t) - \vec{X}_i(t)\right) + \alpha \times (rand(0,1) - 0.5). \tag{1.105}$$

The first term in the position updating formula (1.105) represents the i-th firefly's current position. The second term in (1.105) denotes the change in the position of the firefly at $\vec{X}_i(t)$ due to the attraction toward the brighter firefly at $\vec{X}_j(t)$. Hence it is apparent that the brightest firefly with no more attractive firefly in the current sorted population P_t will have no motion due to the second term and may get stuck at the local optima. To circumvent the problem, the last term is introduced in (1.105) for the random movement of the fireflies with a step-size of $\alpha \in (0, 1)$. Here, *rand*(0,1) is a random number generator uniformly distributed in the range (0, 1). This step is repeated for $i = [1, NP]$. After completion of its journey mediated by the brighter ones, the updated position of the i-th firefly is represented by $\vec{X}_i(t+1)$ for $i = [1, NP]$.

After each evolution, Sections 1.4.2.2 and 1.4.2.3 are repeated until one of the following conditions for convergence is satisfied. These conditions include restraining the number of iterations, maintaining error limits, or the both, whichever occurs earlier. In Algorithm 1.31, the number of iterations is considered as the condition of convergence.

1.4.3 Imperialist Competitive Algorithm

ICA is a population-based stochastic algorithm, which is inspired by the sociopolitical evolution and the imperialistic competitive policy of a government to extend its power beyond its boundaries. It has earned wide popularity because of its noticeable performance in computational optimization with respect to the quality of solutions [89]. Like any other EAs, ICA starts with an initial population of solutions, called countries. The countries are classified into two groups – imperialists and colonies, based on their ruling power (which is inversely proportional to their objective function values). The colonies (weaker countries) with their relevant imperialist (stronger country) form some empires. In each empire, the imperialist pursues an assimilation policy to improve the economy, culture, and political situation of its colonies, thus winning their loyalty. Moreover, the empires take part in the imperialistic competition in an attempt to gain more colonies. In ICA, the assimilation of colonies toward their respective imperialists along with the

Algorithm 1.31 Traditional Firefly Algorithm (FA)

Input: $\vec{X} = (x_1, x_2, \ldots, x_D)$, fitness function $f(X)$; // D dimension of the firefly

Output: $\vec{X}_i$, $i \in [1, n]$;

Initialize: Generate population $\vec{X}_i$, $i \in [1, n]$, $\alpha \in (0, 1)$, $\beta_0 = 1$ and $\gamma \in [0.1, 10]$;

```
While (t <MaxGeneration)
  For k=1 to D
    For i=1 to n
      For j=1 to n
        If f(X_i(t)) < f(X_j(t))
          then Move X_i(t) towards X_j(t) in all D dimensions;
        End If;
        Update x_ik(t+1) = x_ik(t) + βr_ij × [x_jk(t) − x_ik(t)] + α(rand − 0.5);
      End For;
    End For;
  End For;
  Rank the fireflies based on current fitness and find the
  current best one;
End While.
```

competition among empires eventually results in just one empire in the world with all the other countries as colonies of that unique empire. An overview of the main steps of the ICA is presented next.

1.4.3.1 Initialization

ICA starts with a population P_t of NP, D-dimensional countries, $\vec{X}_i(t) = \{x_{i,1}(t), x_{i,2}(t), x_{i,3}(t), ..., x_{i,D}(t)\}$ for $i = [1, NP]$ representing the candidate solutions, at the current generation $t = 0$ by randomly initializing in the range $\left[\vec{X}^{\min}, \vec{X}^{\max}\right]$ where $\vec{X}^{\min} = \{x_1^{\min}, x_2^{\min}, ..., x_D^{\min}\}$ and $\vec{X}^{\max} = \{x_1^{\max}, x_2^{\max}, ..., x_D^{\max}\}$. Thus the d-th component (sociopolitical feature) of the i-th country at $t = 0$ is given by

$$x_{i,d}(0) = x_d^{\min} + rand(0,1) \times \left(x_d^{\max} - x_d^{\min}\right), \tag{1.106}$$

where *rand*(0, 1) is a uniformly distributed random number lying between 0 and 1 and $d = [1, D]$. The objective function value $f\left(\vec{X}_i(0)\right)$ of the country $\vec{X}_i(0)$ is evaluated for $i = [1, NP]$.

1.4.3.2 Selection of Imperialists and Colonies

The population P_0 is sorted in ascending order of $f\left(\vec{X}_i(0)\right)$ for minimization problem with $i = [1, NP]$. The first N countries with less cost function values are selected as imperialists while the remaining $M = NP - N$ countries are declared as colonies. Hence the population individuals are categorized into two groups of countries – imperialists and colonies.

1.4.3.3 Formation of Empires

The empire under the j-th imperialist is constructed based on its ruling power. To accomplish this, first the normalized power of the j-th imperialist country, p_j, is evaluated by (1.107) with $f\left(\vec{X}_{NP}(0)\right)$ representing the objective function value of the weakest country in the current sorted population P_0.

$$p_j = \frac{f\left(\vec{X}_{NP}(0)\right) - f\left(\vec{X}_j(0)\right)}{\sum_{l=1}^{N} f\left(\vec{X}_{NP}(0)\right) - f\left(\vec{X}_l(0)\right)}. \tag{1.107}$$

It is evident from (1.107) that better the j-th imperialist (i.e. less objective function value $f\left(\vec{X}_j(0)\right)$ for minimization problem), higher is the difference $f\left(\vec{X}_{NP}(0)\right) - f\left(\vec{X}_j(0)\right)$ leading to the enhancement of its corresponding ruling power, p_j. Now the initial number of colonies under in the j-th empire, denoted by n_j, is computed by (1.108).

$$n_j = \left\lfloor M \times p_j \right\rfloor, \tag{1.108}$$

such that

$$\sum_{j=1}^{N} n_j = M. \tag{1.109}$$

Here, $\lfloor\ \rfloor$ represents the floor function. According to (1.108), the stronger imperialists with higher ruling power now possess larger empires. Hence p_j symbolizes the fraction of the colonies occupied by the j-th imperialist. Subsequently, the j-th empire is formed by randomly selecting n_j countries from M colonies provided that there will be no common colony between two different empires. Hence the

number of countries within the j-th empire including its imperialist is $n_j + 1$. Let the k-th country belonging to the j-th empire be denoted by $\vec{X}_k^j(t)$ (at generation $t = 0$) for $k = [1, n_j{+}1]$. The countries within the j-th empire are now sorted in ascending order of their objective function values such that the imperialist $\vec{X}_1^j(t)$ in the j-th empire attains the first rank. This step is repeated for $j = [1, N]$.

1.4.3.4 Assimilation of Colonies

Each imperialist country now attempts to improve its empire by enhancing the sociopolitical influences of its colonies. To accomplish this, each country $\vec{X}_k^j(t)$ in the j-th empire now moves toward its corresponding imperialist $\vec{X}_1^j(t)$ by changing its characteristic features following (1.110) for $k = [2, n_j{+}1]$.

$$\vec{X}_k^j(t+1) = \vec{X}_k^j(t) + \beta \times rand(0,1) \times \left(\vec{X}_1^j(t) - \vec{X}_k^j(t)\right). \tag{1.110}$$

Here, $rand(0, 1)$ is a uniformly distributed random number lying between 0 and 1 and β is the assimilation coefficient. The objective function value of the modified colony $f\left(\vec{X}_k^j(t+1)\right)$ is evaluated for $k = [2, n_j{+}1]$. After assimilation, all the countries in the j-th empire are sorted in ascending order of the objective function values and the first ranked country is declared as the imperialist $\vec{X}_1^j(t+1)$ of the same empire for the next generation (i.e. $t = t + 1$). The step is repeated for $j = [1, N]$.

1.4.3.5 Revolution

Revolution creates sudden fluctuation in the economic, cultural, and political aspects of countries in an empire. The colonies in an empire are now equipped with the power of randomly changing their sociopolitical attributes instead of being assimilated by their corresponding imperialist. It resembles the mutation of trial solutions in the traditional EA. The revolution rate η in the algorithm indicates the percentage of colonies in each empire which will undergo the revolution process. A high value of revolution rate therefore fortifies the explorative power at a cost of poor exploitation capability. Hence a moderate value of revolution rate is favored. Revolution is implemented by randomly selecting $\eta \times n_j$ countries (including the imperialist) in the j-th empire (for $j = [1, N]$) and then they are replaced by randomly initialized countries characterized by new sociopolitical nature. After revolution, as in case of assimilation, all the countries in each empire are sorted in ascending order of the objective function values so that its imperialist is at the first position. The step is repeated for all empires.

1.4.3.6 Imperialistic Competition

All the N empires now participate in an imperialistic competition to take possession of colonies of other weaker empires based on their ruling power. The colonies of the weaker empires will be gradually eluded from the ruling power of their corresponding imperialists and will be thereafter controlled by some other stronger empires. Consequently, the weaker empires will be losing their power and ultimately may be eradicated from the competition. The imperialistic competition along with the collapse mechanism will progressively result in an increment in the power of more dominant empires and diminish the power of weaker ones. The imperialistic competition encompasses the following steps.

Total Empire Power Evaluation

Once an empire is constructed under the dominance of the j-th imperialist $\vec{X}_1^j(t+1)$, the power of the respective empire is compositely influenced by the objective function value of $\vec{X}_1^j(t+1)$ as well as the constituent colonies $\vec{X}_k^j(t+1)$ (after assimilation) under the respective j-th empire for $k = [2, n_j{+}1]$. The total objective function value of the j-th empire is evaluated as follows:

$$tc_j = f\left(\vec{X}_1^j(t+1)\right) + \xi \cdot \frac{1}{n_j} \sum_{k=2}^{n_j+1} \vec{X}_k^j(t+1). \tag{1.111}$$

Here, $\xi < 1$ is a positive number which regulates the influence of the constituent colonies to control the ruling power of the empire. A tiny value of ξ causes the total power of the j-th empire to be determined by its imperialist $\vec{X}_1^j(t+1)$ only, while increasing the value of ξ accentuates the importance of the colonies in deciding the total power of the respective empire. The N empires now are sorted in ascending order of tc_j for $j = [1, N]$. Then the normalized possession power of the j-th empire, pp_j, is evaluated by (1.112) with tc_N representing the total objective function value of the weakest empire in the current population P_t.

$$pp_j = \frac{tc_N - tc_j}{\sum_{l=1}^{N} tc_N - tc_l}. \tag{1.112}$$

It is evident from (1.112) that stronger the j-th empire (i.e. less the total objective function value tc_j for minimization problem), higher is the possession power, pp_j, which consecutively increases its probability of seizing colonies from weaker empires. This step is repeated for $j = [1, N]$.

Reassignment of Colonies and Removal of Empire

The empire with least possession power is interpreted as being defeated in the competition. Let the weakest colony of this weakest empire be denoted as $\vec{X}_{worst}$, which is now removed from the dominance of its currently ruling imperialist and reassigned as a new colony to one of the stronger empires based on their possession probabilities. It is noteworthy that $\vec{X}_{worst}$ will not be possessed by the most powerful empires, but stronger the empire, more likely to possess $\vec{X}_{worst}$. To accomplish this, the possession probability of the j-th empire is computed as follows for $j = [1, N]$:

$$prob_j = pp_j - rand(0, 1). \tag{1.113}$$

Now $\vec{X}_{worst}$ is assigned as a new colony to the j-th empire for which the possession probability $prob_j$ is maximum. However, if the worst colony consists of only its imperial before exclusion operation (i.e. $\vec{X}_{worst}$ is the imperialist of the weakest empire), the removal of $\vec{X}_{worst}$ will result in the collapse of the weakest empire.

Union of Empires

The disagreement between two empires may be assessed by the difference in their respective sociopolitical features. This dissimilarity between any two empires, j and l, is evaluated by taking the Euclidean distance between the respective imperialists $\vec{X}_1^j(t+1)$ and $\vec{X}_1^l(t+1)$ as in (1.114) for $j, l = [1, N]$.

$$Dist_{j,l} = \left\| \vec{X}_1^j(t+1) - \vec{X}_1^l(t+1) \right\|. \tag{1.114}$$

If $Dist_{j,l}$ is less than a predefined threshold, Th, the two empires are merged into one empire. The stronger country among $\vec{X}_1^j(t+1)$ and $\vec{X}_1^l(t+1)$ is declared as the imperialist of the newly formed empire.

After each evolution, we repeat from Section 1.4.3.4 until one of the following conditions for convergence is satisfied. Stop criteria include a bound by the number of iterations, achieving a sufficiently low error or aggregations thereof.

1.4.4 Differential Evolution Algorithm

Differential evolution (DE) algorithm is a stochastic, population-based global optimization algorithm, introduced by [133], to optimize real parameter, real-valued functions [48].

1.4.4.1 Initialization

Range of each parameter, i.e. the upper and lower boundaries for each parameter, is defined, and then randomly these parameters are initialized.

1.4.4.2 Mutation

The step mutation expands the search-space. Mutation is done by (1.115), where $F \in [0, 2]$ is the mutation factor. $x_{r_{1,G}}, x_{r_{2,G}}$ and $x_{r_{3,G}}$ are the randomly selected variables with i, r_1, r_2, r_3 and G are index. $v_{i,G+1}$ refers to the donor vector.

$$v_{i,G+1} = x_{r_{1,G}} + F\left(x_{r_{2,G}} - x_{r_{3,G}}\right). \tag{1.115}$$

1.4.4.3 Recombination

Employing the target vector $x_{i,G}$ and the elements of the donor vector $v_{i,G+1}$, the trial solution vector $u_{i,G+1}$ is evaluated by following (1.116).

$$u_{i,j,G+1} = \begin{cases} v_{i,j,G+1} & \text{if rand}() \leq \text{CR or } j = I_{rand} \\ x_{i,j,G} & \text{if rand}() > \text{CR or } j \neq I_{rand} \end{cases}, \tag{1.116}$$

where $i = [1, N]$, $j = [1, D]$ and $v_{i,G+1} \neq x_{i,\,G}$ is checked by I_{rand}.

1.4.4.4 Selection

The target solution $x_{i,G}$ is compared with the trial solution vector $u_{i,G+1}$ and the next generation is selected by (1.117).

$$x_{i,G+1} = \begin{cases} u_{i,G+1} & \text{if } f(u_{i,G+1}) \leq f(x_{i,G}) \\ x_{i,G} & \text{otherwise} \end{cases}, \tag{1.117}$$

where $i = [1, N]$.

Sections 1.4.4.2 to 1.4.4.4 continue until the termination criterion as explained earlier is reached.

1.4.5 Off-line Optimization

By SI and EO algorithms only the off-line optimization is possible due to their huge run-time complexity. In case of multi-robot coordination, robots evaluate the optimal trajectory (collection of coordinates) off-line in the sense of system recourse (time and/or energy) utilization. After off-line optimization of the trajectory, it is executed in the real-robot.

1.4.6 Performance Analysis of Optimization Algorithms

The performance of SI, EA, and their hybridization can be analyzed by the following performance metrics. *Quality* of s*olution* within a fixed epoch and the

convergence time are two performance matrices of the SI and EA. In addition, mean best objective function versus function evaluation, accuracy versus function evaluation, and function evaluation versus search space dimensionality can be considered as the performance metrics. In spite of the abovementioned performance metrics, statistical test is conducted over the algorithms for performance measurement.

1.4.6.1 Friedman Test

Friedman test [64], which is a nonparametrical statistical test, may be carried out on the average objective function values of each of the algorithms for fixed independent runs, assuming a fixed dimension. To carry out the Friedman test, first the average ranking (R_i) for each of the considered algorithms is calculated as the mean of the individual ranks obtained by them over all the considered N number of benchmark functions, as shown in (1.118),

$$R_i = \frac{1}{N}\sum_{j=1}^{N} r_i^j. \tag{1.118}$$

Here, r_i^j refers to the individual rank attained by the i-th algorithm for the j-th benchmark function and the results have been computed considering N benchmark functions. In the next step, a term formally defining the Friedman statistic, which follows a χ_F^2 distribution with $(k-1)$ degrees of freedom, has been evaluated using (1.119),

$$\chi_F^2 = \frac{12N}{k(k+1)}\left[\sum_{i=1}^{k} R_i^2 - \frac{k(k+1)^2}{4}\right]. \tag{1.119}$$

1.4.6.2 Iman–Davenport Test

Moreover, Iman–Davenport test [65] can also been conducted in order to substantiate the findings of the former statistical analysis. It is basically a deviation from the Friedman test producing more precise results and the Iman–Davenport statistics is calculated as follows:

$$F_F = \frac{(N-1)\times\chi_F^2}{N\times(k-1)-\chi_F^2}. \tag{1.120}$$

Tabular analysis can be shown in case of Friedman test, which demonstrates that the null hypothesis has been rejected if the computed value of χ_F^2 is greater than the critical value of the χ_F^2 distribution with degrees of freedom $(k-1)$ at probability of α ($\chi_{3,\alpha}^2$). For, Iman–Davenport test, the statistic is distributed with $(k-1)$ and $(k-1)\times(N-1)$ degrees of freedom. Likewise, the null hypothesis has been

rejected as the calculated value of F_F is greater than the critical value of the F_F distribution with degrees of freedom $(k-1)$ and $(k-1)\times(N-1)$ at probability of α ($F_{(k-1),(N-1),\alpha}$). It is obvious that the proposed algorithm is the most efficient one, hence, in the post-hoc analysis, the proposed algorithm is assumed to be the control method.

For multi-robot trajectory (path) planning, Average total path deviation, Average Uncovered Target Distance, Average total path traversed, and numbers of steps required are considered as the performance metrics.

1.5 Summary

This chapter introduces multi-robot coordination algorithms for complex real-world problems employing the principles of RL, GT, DP, and/or EA. As expected, this chapter includes a thorough survey of the exiting literature of RL with a brief overview of the EO to examine the role of the algorithms in view of the multi-agent coordination. Here, multi-robot coordination is achieved by employing the EO, and specially RL for cooperative, competitive, and their composition for application to static and dynamic games. The remainder of the chapter provides an overview of the metrics used to compare the performance of the algorithms while coordinating.

References

1 Arkin, R.C. (1998). *Behavior-Based Robotics*. MIT Press.

2 Sen, S., Sekaran, M., and Hale, J. (1994). Learning to coordinate without sharing information. *Proceedings of the American Association for Artificial Intelligence* **94**: 426–431.

3 Kapetanakis, S. and Kudenko, D. (2002). Reinforcement learning of coordination in cooperative multi-agent systems. *Proceedings of the American Association for Artificial Intelligence* **18**: 326–331.

4 Konar, A., Chakraborty, I.G., Singh, S.J. et al. (2013). A deterministic improved Q-learning for path planning of a mobile robot. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* **43** (5): 1141–1153.

5 Sadhu, A.K., Rakshit, P., and Konar, A. (2016). A modified Imperialist Competitive Algorithm for multi-robot stick-carrying application. *Robotics and Autonomous Systems* **76**: 15–35.

6 Stentz, A. (1997). Optimal and efficient path planning for partially known environments. In: *Intelligent Unmanned Ground Vehicles*, vol. **388** (eds. M. Hebert and C. Thorpe), 203–220. Boston, MA: Springer.

7 Xu, X., Zuo, L., and Huang, Z. (2014). Reinforcement learning algorithms with function approximation: recent advances and applications. *Information Sciences* **261**: 1–31.

8 Buşoniu, L., Lazaric, A., Ghavamzadeh, M. et al. (2012). Least-squares methods for policy iteration. In: *Reinforcement Learning* (eds. M. Wiering and M. van Otterlo), 75–109. Berlin Heidelberg: Springer.

9 Xu, X., Hu, D., and Lu, X. (2007). Kernel-based least squares policy iteration for reinforcement learning. *IEEE Transactions on Neural Networks* **18** (4): 973–992.

10 Golub, G. and Kahan, W. (1965). Calculating the singular values and pseudo-inverse of a matrix. *Journal of the Society for Industrial and Applied Mathematics, Series B: Numerical Analysis* **2** (2): 205–224.

11 Lagoudakis, M.G. and Parr, R. (2003). Least-squares policy iteration. *The Journal of Machine Learning Research* **4**: 1107–1149.

12 Martins, M.F. and Demiris, Y. (2010). Learning multirobot joint action plans from simultaneous task execution demonstrations. *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems* **1**: 931–938.

13 Cao, Y.U., Fukunaga, A.S., and Kahng, A.B. (1997). Cooperative mobile robotics: antecedents and directions. *Autonomous Robots* **4** (1): 7–27.

14 Farinelli, A., Iocchi, L., and Nardi, D. (2004). Multi-robot systems: a classification focused on coordination. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics* **34** (5): 2015–2028.

15 Szer, D., Charpillet, F., and Zilberstein, S. (2005). MAA∗: a heuristic search algorithm for solving decentralized POMDPs. *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence-UAI*, Edinburgh, Scotland (26–29 July 2005).

16 Dias, M.B., Zlot, R., Kalra, N., and Stentz, A. (2006). Market-based multirobot coordination: a survey and analysis. *Proceedings of the IEEE* **94** (7): 1257–1270.

17 Stentz, A. and Dias, M.B. (1999). A Free Market Architecture for Coordinating Multiple Robots. Technical Report, CMU-RI-TR-99-42, Robotics Institute, Carnegie Mellon University.

18 Dias, M.B. and Stentz, A. (2002). Opportunistic optimization for market-based multirobot control. *IEEE/RSJ International Conference on Intelligent Robots and Systems* **3**: 2714–2720.

19 Dias, M.B. (2004). Traderbots: a new paradigm for robust and efficient multirobot coordination in dynamic environments. Doctoral dissertation, Carnegie Mellon University Pittsburgh.

20 Sandholm, T. (2002). Algorithm for optimal winner determination in combinatorial auctions. *Artificial Intelligence* **135** (1): 1–54.

21 Berhault, M., Huang, H., Keskinocak, P. et al. (2003). Robot exploration with combinatorial auctions. *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems, (IROS 2003)* **2**: 1957–1962.

22 Dias, M.B., Zlot, R., Zinck, M. et al. (2004). A versatile implementation of the TraderBots approach for multirobot coordination. *Proceedings of the 8th Conference on Intelligent Autonomous Systems (IAS-8)*, Amsterdam, Netherlands (10–12 March 2004).

23 Badreldin, M., Hussein, A., and Khamis, A. (2013). A comparative study between optimization and market-based approaches to multi-robot task allocation. *Advances in Artificial Intelligence* **2013**: 1–11.

24 Konar, A., Chakraborty, I.G., Singh, S.J. et al. (2013). A deterministic improved Q-learning for path planning of a mobile robot. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* **43** (5): 1–13.

25 Marden, J.R., Arslan, G., and Shamma, J.S. (2009). Cooperative control and potential games. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics* **39** (6): 1393–1407.

26 Fax, A. and Murray, R.M. (2004). Information flow and cooperative control of vehicle formations. *IEEE Transactions on Automation Control* **49**: 1465–1476.

27 Kashyap, A., Başar, T., and Srikant, R. (2006). Consensus with quantized information updates. *Proceedings of the 45th IEEE Conference on Decision and Control*, San Diego, CA (13–15 December 2006).

28 Olfati-Saber, R., Fax, A., and Murray, R.M. (2007). Consensus and cooperation in networked multi-agent systems. *Proceedings of the IEEE* **95** (1): 215–233.

29 Mohanty, M., Mishra, A., and Routray, A. (2009). A non-rigid motion estimation algorithm for yawn detection in human drivers. *International Journal of Computational Vision and Robotics* **1** (1): 89–109.

30 LaValle, S.M. (2006). *Planning Algorithms*. Cambridge university press.

31 Nilsson, N.J. (2014). *Principles of Artificial Intelligence*. Morgan Kaufmann.

32 Konar, A. (1999). *Artificial Intelligence and Soft Computing: Behavioral and Cognitive Modeling of the Human Brain*. CRC Press.

33 Bhattacharya, P. and Gavrilova, M.L. (2008). Roadmap-based path planning: using the Voronoi diagram for a clearance-based shortest path. *IEEE Robotics and Automation Magazine* **15** (2): 58–66.

34 Gayle, R., Moss, W., Lin, M.C., and Manocha, D. (2009). Multi-robot coordination using generalized social potential fields, *Proceedings of the IEEE International Conference on Robotics and Automation*. Kobe, Japan (12–17 May 2009), pp. 106–113.

35 Sutton, R.S. and Barto, A.G. (1998). *Reinforcement Learning: An Introduction*. Cambridge, MA: The MIT Press.

36 Krishna, K.M. and Hexmoor, H. (2004). Reactive collision avoidance of multiple moving agents by cooperation and conflict propagation. *IEEE International Conference on Robotics and Automation (ICRA)* **3**: 2141–2146.

37 Farinelli, A., Iocchi, L., and Nardi, D. (2004). Multirobot systems: a classification focused on coordination. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* **34** (5): 2015–2028.

38 Myerson, R.B. (1991). *Game Theory: Analysis of Conflict*. Cambridge: Harvard University Press.

39 Brown, G.W. (1951). Iterative solution of games by fictitious play. *Activity Analysis of Production and Allocation* **13** (1): 374–376.

40 Kandori, M., Mailath, G.J., and Rob, R. (1993). Learning, mutation, and long run equilibria in games. *Econometrica: Journal of the Econometric Society* **61**: 29–56.

41 Neumann, L.J. and Morgenstern, O. (1947). *Theory of games and economic behavior*, vol. **60**. Princeton: Princeton University Press.

42 Nash, J. (1951). Non-cooperative games. *Annals of Mathematics* **54**: 286–295.

43 Owen, G. (1995). *Game Theory*. Academic Press.

44 Basar, T. and Olsder, G.J. (1999). *Dynamic Noncooperative Game Theory*, vol. **23**. SIAM.

45 Fudenberg, D. and Kreps, D.M. (1992). *Lectures on Learning and Equilibrium in Strategic Form Games*. Louvain-La-Neuve: Core Foundation.

46 Howard, R.A. (1960). *Dynamic Programming and Markov Processes*. The MIT Press.

47 Bellman, R.E. (1957). Dynamic programming. *Proceedings of the National Academy of Science of the United States of America* **42** (10): 34–37.

48 Yang, X.S. (2009). Firefly algorithms for multimodal optimization, stochastic algorithms: foundations and applications. *SAGA, Lecture Notes in Computer Sciences* **5792**: 169–178.

49 Narimani, R. and Narimani, A. (2013). A new hybrid optimization model based on imperialistic competition and differential evolution meta-heuristic and clustering algorithms. *Applied Mathematics in Engineering, Management and Technology* **1** (2): 1–9.

50 Subudhi, B. and Jena, D. (2011). A differential evolution based neural network approach to nonlinear system identification. *Applied Soft Computing* **11** (1): 861–871.

51 Ramezani, F., Lotfi, S., and Soltani-Sarvestani, M.A. (2012). A hybrid evolutionary imperialist competitive algorithm (HEICA). In: *Intelligent Information and Database Systems*, Part I, LNAI, vol. **7196** (eds. J.-S. Pan, S.-M. Chen and N.T. Nguyen), 359–368. Berlin Heidelberg: Springer.

52 Khorani, V., Razavi, F., and Ghoncheh, A. (2010). *A New Hybrid Evolutionary Algorithm Based on ICA and GA: Recursive-ICA-GA*, 131–140. IC-AI.

53 Nozarian, S. and Jahan, M.V. (2012). A Novel Memetic Algorithm with Imperialist Competition as Local Search. *International Proceedings of Computer Science and Information Technology* **30**: 54–59.

54 Lin, J.L., Tsai, Y.H., Yu, C.Y., and Li, M.S. (2012). Interaction enhanced imperialist competitive algorithms. *Algorithms* **5** (4): 433–448.

55 Coelho, L.D.S., Afonso, L.D., and Alotto, P. (2012). A modified imperialist competitive algorithm for optimization in electromagnetic. *IEEE Transactions on Magnetics* **48** (2): 579–582.

56 Bidar, M. and Rashidy, H.K. (2013). Modified firefly algorithm using fuzzy tuned parameters. *Proceedings of the 13th Iranian Conference on Fuzzy Systems (IFSC), IEEE*, Iran Qazvin (27–29 August 2013), pp. 1–4.

57 Seuken, S. and Zilberstein, S. (2007). Memory-Bounded Dynamic Programming for DEC-POMDPs. *IJCAI*, pp. 2009–2015.

58 Seuken, S. and Zilberstein, S. (2012). Improved memory-bounded dynamic programming for decentralized POMDPs. *arXiv preprint arXiv*. pp. 1206.5295.

59 K. Alton and I. M. Mitchell, Efficient dynamic programming for optimal multi-location robot rendezvous. *Proceedings of the 47th IEEE Conference on Decision and Control*, MEX Cancun (9–11 December 2008), pp. 2794–2799.

60 Nudelman, E., Wortman, J., Shoham, Y., and Leyton-Brown, K. (2004). Run the GAMUT: a comprehensive approach to evaluating game-theoretic algorithms. *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems* **2**: 880–887.

61 Kennedy, J., Eberhart, R., and Shi, Y. (2001). *Swarm Intelligence*. Los Altos, CA: Morgan Kaufmann.

62 Das, S., Abraham, A., and Konar, A. (2008). Particle swarm optimization and differential evolution algorithms: technical analysis, applications and hybridization perspectives. In: *Advances of Computational Intelligence in Industrial Systems*, 1–38. Berlin Heidelberg: Springer.

63 Gargari, E.A. and Lucas, C. (2007). Imperialist competitive algorithm: an algorithm for optimization inspired by imperialistic competition. *Proceedings of the IEEE Congress in Evolutionary Computation, CEC*, Singapore (25–28 September 2007), pp. 4661–4667.

64 Horng, M.H. and Jiang, T.W. (2010). The codebook design of image vector quantization based on the firefly algorithm. In: *International Conference on Computational Collective Intelligence*, Part III, LNAI, vol. **6423** (eds. J.-S. Pan, S.-M. Chen and N.T. Nguyen), 438–447. Berlin, Heidelberg: Springer.

65 Abidin, Z.Z., Arshad, M.R., and Ngah, U.K. (2011). A simulation based fly optimization algorithm for swarms of mini-autonomous surface vehicles application. *Indian Journal of Marine Sciences* **40** (2): 250–266.

66 Belal, M., Gaber, J., El-Sayed, H., and Almojel, A. (2006). Swarm intelligence. In: *Handbook of Bioinspired Algorithms and Applications*, CRC Computer and Information Science, vol. **7** (eds. S. Olariu and A.Y. Zomaya). Chapman and Hall.

67 M. Dorigo, *In The Editorial of the First Issue of: Swarm Intelligence Journal*, Springer Science + Business Media, LLC, Vol.**1**, No. 1, pp. 1–2, 2007.

68 Hosseini, S. and Al Khaled, A. (2014). A survey on the Imperialist Competitive Algorithm metaheuristic: implementation in engineering domain and directions for future research. *Applied Soft Computing* **24**: 1078–1094.

69 Pugh, J., Zhang, Y., and Martinoli, A. (2005). Particle swarm optimization for unsupervised robotic learning. *Proceedings of the Swarm Intelligence Symposium*, Pasadena, CA (June 2005), pp. 92–99.

70 Pugh, J. and Martinoli, A. (2006). Multi-robot learning with particle swarm optimization. *International Proceedings of the Autonomous Agents and Multi-agent Systems*, Japan (8–12 May 2006), pp. 441–448.

71 Hu, J. and Wellman, M.P. (2003). Nash Q-learning for general-sum stochastic games. *The Journal of Machine Learning Research* **4**: 1039–1069.

72 Greenwald, A., Hall, K., and Serrano, R. (2003). Correlated Q-learning. *Proceedings of the International Conference on Machine Learning* **3**: 242–249.

73 Kaelbling, L.P., Littman, M.L., and Moore, A.W. (1996). Reinforcement learning: a survey. *Journal of Artificial Intelligence Research* **4**: 237–285.

74 Dijkstra, E.W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik* **1**: 269–271.

75 Najnin, S. and Banerjee, B. (2018). Pragmatically framed cross-situational noun learning using computational reinforcement models. In: *Frontiers in Psychology (Cognitive Science Section)*, vol. **9**, Article 5 (ed. J.L. McClelland).

76 Fraternali, F., Balaji, B., and Gupta, R. (2018). Scaling configuration of energy harvesting sensors with reinforcement learning. *Proceedings of the 6th International Workshop on Energy Harvesting and Energy-Neutral Sensing Systems*, Shenzhen, China (4 November 2018), pp. 7–13. ACM.

77 Lawhead, R.J. and Gosavi, A. (2019). A bounded actor–critic reinforcement learning algorithm applied to airline revenue management. *Engineering Applications of Artificial Intelligence* **82**: 252–262.

78 Schawartz, H.M. (2014). *Multi-Agent Machine Learning a Reinforcement Approach*. Wiley.

79 Berry, D. and Fristedt, B. (1985). *Bandit Problems*. Chapman and Hall.

80 Filar, J. and Vrieze, K. (2012). *Competitive Markov Decision Processes*. Springer Science & Business Media.

81 Claus, C. and Boutilier, C. (1998). The dynamics of reinforcement learning in cooperative multiagent systems. *Proceedings of the National Conference on Artificial Intelligence* **15**: 746–752.

82 Jain, R. and Varaiya, P. (2010). Simulation-based optimization of Markov decision processes: an empirical process theory approach. *Automatica* **46** (8): 1297–1304.

83 Kemeny, J.G. and Laurie Snell, J. (1960). *Finite Markov Chains*. New York, Berlin, Tokyo: Springer-Verlag.

84 Barto, A.G., Sutton, R.S., and Watkins, C.J. (1989). *Learning and Sequential Decision Making*. Amhers: University of Massachusetts.

85 Busoniu, L., Babuska, R., and De Schutter, B. (2008). A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews* **38** (2): 156–172.

86 Young, H.P. (1993). The evolution of conventions. *Econometrica: Journal of the Econometric Society* **61** (1): 57–84.

87 Fudenberg, D. and Levine, D.K. (1993). Steady state learning and Nash equilibrium. *Econometrica: Journal of the Econometric Society* **61**: 547–573.

88 Kalai, E. and Lehrer, E. (1993). Rational learning leads to Nash equilibrium. *Econometrica: Journal of the Econometric Society* **61**: 1019–1045.

89 Singh, S., Jaakkola, T., Littman, M.L., and Szepesvari, C. (1998). Convergence results for single-step on-policy reinforcement learning algorithms. *Machine Learning* **38** (3): 287–308.

90 Tan, M. (1993). Multi-agent reinforcement learning: independent vs. cooperative agents. *Proceedings of the Tenth International Conference on Machine Learning*, Amherst (27–29 June 1993), pp. 330–337.

91 Lauer, M. and Riedmiller, M. (2000). An algorithm for distributed reinforcement learning in cooperative multi-agent systems. *Proceedings of the Seventeenth International Conference on Machine Learning*, Stanford, CA (29 June to 2 July 2000).

92 Littman, M.L. (2001). Value-function reinforcement learning in Markov games. *Cognitive Systems Research* **2** (1): 55–66.

93 Littman, M.L. and Szepesvári, C. (1996). A generalized reinforcement-learning model: convergence and applications. *Proceedings of the International Conference on Machine Learning* **13**: 310–318.

94 Szepesvári, C. and Littman, M.L. (1999). A unified analysis of value-function-based reinforcement-learning algorithms. *Neural Computation* **11** (8): 2017–2060.

95 Hu, J. and Wellman, M.P. (1998). Multiagent reinforcement learning: theoretical framework and an algorithm. *Proceedings of the International Conference on Machine Learning* **98**: 242–250.

96 Wang, X. and Sandholm, T. (2002). Reinforcement learning to play an optimal Nash equilibrium in team Markov games. *Advances in Neural Information Processing Systems* **2**: 1571–1578.

97 Kok, J.R. and Vlassis, N. (2004). Sparse cooperative Q-learning. *Proceedings of the International Conference on Machine Learning*, Banff, Alberta (4–8 July 2004).

98 Wang, Y. and de Silva, C.W. (2008). A machine learning approach to multi-robot coordination. *Engineering Application of Artificial Intelligence* **21**: 470–484.

99 Zhang, Z., Zhao, D., Gao, J. et al. (2016). FMRQ: a multiagent reinforcement learning algorithm for fully cooperative tasks. *IEEE Transactions on Cybernetics* **47**: 2168–2267.

100 Littman, M.L. (1994). Markov games as a framework for multi-agent reinforcement learning. *Proceedings of the Eleventh International Conference on Machine Learning* **157**: 157–163.

101 Bianch, R.A.C., Martins, M.F., Ribeiro, C.H.C., and Costa, A.H.R. (2014). Heuristically: accelerated multiagent reinforcement learning. *IEEE Transactions on Cybernetics* **44** (2): 252–265.

102 Bianchi, R.A.C., Ribeiro, C.H.C., and Costa, A.H.R. (2008). Accelerating autonomous learning by using heuristic selection of actions. *Journal Heuristics* **14** (2): 135–168.

103 Bianchi, R.A.C. (2012). Heuristically accelerated reinforcement learning: theoretical and experimental results. *Frontiers in Artificial Intelligence and Applications* **242**: 169–174.

104 Bianchi, R.A.C. (2007). Heuristic selection of actions in multiagent reinforcement learning. *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, Hyderabad, India (6–12 January 2007), pp. 690–695.

105 Conitzer, V. (2009). Approximation guarantees for fictitious play. *Proceedings of the 47th Annual Allerton Conference on Communication, Control, and Computing, IEEE*, Allerton House, IL (30 September to 2 October 2009), pp. 636–643.

106 Powers, R. and Shoham, Y. (2004). New criteria and a new algorithm for learning in multi-agent systems, *Advances in Neural Information Processing Systems*, Vancouver, British Columbia (13–18 December 2004), pp. 1089–1096.

107 Bowling, M. and Veloso, M. (2002). Multiagent learning using a variable learning rate. *Artificial Intelligence* **136** (2): 215–250.

108 Conitzer, V. and Sandholm, T. (2007). AWESOME: a general multiagent learning algorithm that converges in self-play and learns a best response against stationary opponents. *Machine Learning* **67** (1–2): 23–43.

109 Stone, P. and Veloso, M. (2000). Multiagent systems: a survey from a machine learning perspective. *Autonomous Robots* **8** (3): 345–383.

110 Tesauro, G. (2003). Extending Q-learning to general adaptive multi-agent systems. *Advances in Neural Information Processing Systems,* Vancouver and Whistler, British Columbia (8–13 December 2003), pp. 871–878.

111 Singh, S., Kearns, M., and Mansour, Y. (2000). Nash convergence of gradient dynamics in general-sum games. *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, San Francisco, CA (30 June to 3 July 2000), pp. 541–548. Morgan Kaufmann Publishers Inc.

112 Reinhard, H. (1986). *Differential Equations: Foundations and Applications*. North Oxford: Academic.

113 Boyd, S. and Vandenberghe, L. (2004). *Convex Optimization*. Cambridge University Press.

114 Friedman, J., Hastie, T., and Tibshirani, R. (2001). *The Elements of Statistical Learning*, Springer series in statistics, vol. **1**. Berlin: Springer.

115 Boser, B.E., Guyon, I.M., and Vapnik, V.N. (1992). A training algorithm for optimal margin classifiers. *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, Pittsburgh, PA (27–29 July 1992), pp. 144–152. ACM.

116 Bansal, N., Blum, A., Chawla, S., and Meyerson, A. (2003). Online oblivious routing. *Proceedings of the Fifteenth Annual ACM Symposium on Parallel Algorithms and Architectures*, San Diego, CA, (7–9 June 2003), pp. 44–49. ACM.

117 Boot, J.C. (1964). *Quadratic Programming: Algorithms, Anomalies, Applications*. Rand McNally.

118 Zinkevich, M. (2003). Online convex programming and generalized infinitesimal gradient ascent. *Proceedings of the International Conference on Machine Learning*, Washington, DC (21–24 August 2003).

119 Bowling, M. (2005). Convergence and no-regret in multiagent learning. *Advances in Neural Information Processing Systems* **17**: 209–216.

120 Könönen, V. (2004). Asymmetric multiagent reinforcement learning. *Web Intelligence and Agent Systems: An International Journal* **2** (2): 105–121.

121 Littman, M.L. (2001). Friend-or-foe Q-learning in general-sum games. *Proceedings of the International Conference on Machine Learning* **1**: 322–328.

122 Hu, Y., Gao, Y., and An, B. (2015). Multiagent reinforcement learning with unshared value functions. *IEEE Transactions on Cybernetics* **45** (4): 647–661.

123 Hu, Y., Gao, Y., and An, A. (2015). Accelerating multiagent reinforcement learning by equilibrium transfer. *IEEE Transactions on Cybernetics* **45** (7): 1289–1302.

124 Bowling, M. and Veloso, M. (2001). Rational and convergent learning in stochastic games. *International Joint Conference on Artificial Intelligence* **17** (1): 1021–1026, Lawrence Erlbaum Associates Ltd.

125 Weiß, G. (1995). *Adaptation and Learning in Multi-agent Systems, Some Remarks and a Bibliography*, 1–21. Berlin, Heidelberg: Springer.

126 Banerjee, B. and Peng, J. (2003). Adaptive policy gradient in multiagent learning. *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, Melbourne, Victoria (14–18 July 2003), pp. 686–692. ACM.

127 Banerjee, B. and Peng, J. (2002). Convergent gradient ascent in general-sum games. In: *Machine Learning: ECML* (eds. T. Elomaa, H. Mannila and H. Toivonen), 1–9. Berlin, Heidelberg: Springer.

128 Weinberg, M. and Rosenschein, J.S. (2004). Best-response multiagent learning in non-stationary environments. *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems* **2**: 506–513.

129 Shoham, Y., Powers, R., and Grenager, T. (2003). Multi-agent Reinforcement Learning: A Critical Survey. Technical Report, Computer Science Department, Stanford University, Stanford.

130 Hu, J. (2003). Best-response algorithm for multiagent reinforcement learning. *Proceedings of the International Conference on Machine Learning*, Washington, DC (21–24 August 2003).

131 Suematsu, N. and Hayashi, A. (2002). A multiagent reinforcement learning algorithm using extended optimal response. *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems: Part 1,* Bologna Italy (July 2002), pp. 370–377. ACM.

132 Cortés-Antonio, P., Rangel-González, J., Villa-Vargas, L.A., et al. (2014). Design and implementation of differential evolution algorithm on FPGA for double-precision floating-point representation. *Acta Polytechnica Hungarica* **11** (4): 139–153.

133 Storn, R. and Price, K., et al. (1996). Minimizing the real functions of the ICEC'96 contest by differential evolution, In Proceedings of IEEE international conference on evolutionary computation. pp 842–844.