Aggregate Reinforcement Learning for Multi-Agent Territory Division: The Hide-and-Seek Game

Mohamed K. Gunady[1], Walid Gomaa[2,*], Ikuo Takeuchi[2,3]
[1] University of Maryland, College Park, Maryland, USA
`mgunady@cs.umd.edu`
[2] Egypt-Japan University of Science and Technology, New Borg El-Arab, Alexandria, Egypt
`walid.gomaa@ejust.edu.eg`
[3] Faculty of Science and Engineering, Waseda University, Tokyo, Japan
`nue@nue.org`

Abstract

In many applications in Robotics such as disaster rescuing, mine detection, robotic surveillance and warehouse systems, it is crucial to build multi-agent systems (MAS) in which agents cooperate to complete a sequence of tasks. For better performance in such systems, e.g. minimizing duplicate work, agents need to agree on how to divide and plan that sequence of tasks among themselves. This paper targets the problem of territory division in the children's game of Hide-and-Seek as a test-bed for our proposed approach. The problem is solved in a hierarchical learning scheme using Reinforcement Learning (RL). Based on Q-learning, our learning model is presented in detail; definition of composite states, actions, and reward function to deal with multiple agent learning. In addition, a revised version of the standard updating rule of the Q-learning is proposed to cope with multiple seekers. The model is examined on a set of different maps, on which it converges to the optimal solutions. After the complexity analysis of the algorithm, we enhanced it by using state aggregation (SA) to alleviate the state space explosion. Two levels of aggregation are devised: topological and hiding aggregation. After elaborating how the learning model is modified to handle the aggregation technique, the enhanced model is examined by some experiments. Results indicate promising performance with higher convergence rate and up to 10x space reduction.

Keywords:

multi-agent systems, reinforcement learning, hierarchical learning, Q-learning, state aggregation, Hide-and-Seek

1 Introduction

Most of the real life applications of multi-agent systems and machine learning consist of highly complicated tasks. One learning paradigm for handling such complexity is to learn simpler tasks first then increase the task complexity in a hierarchical manner till reaching the desired complexity level, similar to the learning paradigm followed by children. Various practical planning problems can be reduced to Hide-and-Seek games in which children learn the basic concepts of path planning, map building, navigation, and team cooperation. Robotics applications in various domains such as disaster rescuing, criminal pursuit, and mine detection/demining are typical examples. One important aspect when dealing with cooperative seekers (in Hide-and-Seek games) is how to divide the search environment to achieve an optimal seeking performance. This can be viewed as a *territory division* problem.

The game of Hide-and-Seek is a simple test-bed for such applications. We adopt this simple test-bed to tackle the task of territory division that frequently appears in multi-robot systems like in security and rescuing robots. Such class of problems that have multiple agents searching for a target resides in an unknown place, but have a likelihood of their possible location. Consider a simple form of the Hide-and-Seek game in which there are multiple seekers and a single hider. Before the game starts, the hider chooses a hiding place based on a pre-determined hiding probability distribution over the environment. Afterwards, it is the task of the seekers to search for the hider and find his hiding place in the minimum amount of time. A game terminates as soon as one seeker finds the hider by reaching his location. Seekers should not just

[*] Currently on leave from the Faculty of Engineering, Alexandria University, Egypt.

scan the whole environment for the hider; otherwise they suffer from a poor seeking performance. In contrast, they should consider only the possible hiding places. A possible hiding place is a location in the environment with a non-zero probability of being used by the hider as a hiding place. Given the hiding probability distribution, the seekers should do an optimized scan that minimizes the expected time of finding the hider. For a game with multiple seekers, a good seeking strategy is to cooperate to apply the concept of dividing the seekers' territories in order to increase the seeking efficiency and thus decrease the game duration. Nevertheless, such cooperation requires physical communication channel among the seekers. In this work, it is assumed that full communication is provided in the learning phase, whilst seekers can work without communication (except that for game termination) in the operation/test phase.

Consider a Hide-and-Seek game with two seekers located at the same position and ready to start seeking. Although the two seekers can go in different directions, there is no guarantee of what scenario may happen in the future. Before they meet once again, there is a high possibility that one seeker will visit areas already visited by the other seeker since the last meeting. This duplication of effort harms the overall seeking performance of the team. The two seekers need to agree on some strategy that gives each seeker a hint about the expected actions of the other seeker, in order to be taken into consideration while planning his next actions. Such strategy can be obtained by dividing the workload between the two seekers. Typically, the map can be divided into territories and one or more of such territories can be assigned to each seeker. A seeker should stick to his seeking territory until finishing, and then it should work on the neighboring territories to help the other seekers with their unfinished work. Some heuristics can be developed for such scheme, but it gives no guarantee of how good the performance would be.

In this paper, we introduce a novel solution for the problem of territory division on Hide-and-Seek games using reinforcement learning (RL). We propose a hierarchical learning scheme to solve the problem of territory division. We present the state, action, and reward function definitions. In addition, a revised version of the standard updating rule of *Q*-learning is proposed to handle multiple seekers. The proposed approach is examined on a set of different maps and resulted in convergence to the optimal solution in all cases. However, RL suffers from the curse of dimensionality with the problem space. Hence, finally we tackle this problem using state aggregation (SA). The idea is to group similar states into one aggregate state, and then do learning over the aggregate level, which is significantly reduced in size. We enhance the learning model using two levels of aggregation, topological aggregation and hiding aggregation. The enhanced model is examined and the results indicate that the computational resources (time and space) and the amount of experience tuples (needed for learning) are highly reduced. There was some preliminary work done in Gunady et al. (2012). However, in this paper we thoroughly discuss the territory division problem and its related literature. The proposed learning model is aided with complexity analysis of the algorithm for performance evaluation. We enhance the learning model using aggregation techniques that show promising results.

Section 1 is an introduction. Section 2 gives an overview of the different approaches to solve the territory division problem. It provides a review of the related work done on similar problems using combinatorial approaches. Section 3 outlines the necessary details of *Q*-learning, the RL algorithm adopted in this work, and the SA concept. Section 4 illustrates the proposed hierarchical learning scheme along with the experimental results. An analysis of the state space complexity is then discussed. In Section 5 we tackle the problem of tremendous time and space complexity by means of SA. It shows how the hierarchical learning scheme is extended to include the use of SA. The enhanced model is examined and shows promising results. Finally, Section 6 concludes the paper.

2 Related Work

The closest related work to the territory division problem is the work done in the traveling salesman problem (TSP). TSP has been studied since the 18th century and research has made significant developments on it since then (Schrijver 1960). Given a set of connected cities, the TSP problem is to find the best visiting strategy to traverse all the cities and return back to the starting point with the minimum cost (travel distance). With a brute-force computational complexity of $O(n!)$, it is a hard combinatorial optimization problem which is classified as *NP-hard*. TSP plays a crucial role in operations research and theoretical computer science due to the various applications of this problem, e.g. vehicle routing problem

(VRP), computer wiring, and many scheduling problems (Balas and Toth, 1985; Gilbert and Hofstra 1992; Lenstra and Rinnooy Kan 1974; Ralphs 2003). The territory division problem seems similar to a class of TSP problems, namely *multiple Travelling Salesman Problem* (mTSP). In mTSP there are *m* salesmen at the same starting city, and it is required to find a route for each salesman starts and ends at the same city so that every city (except the starting city) is visited by a salesman exactly once and the total traveling cost is minimized (Bektas 2006).

TSP has a huge literature and different solution approaches. It is typically modeled as a weighted graph. The literature on TSP is divided into: (1) algorithms to find exact solutions, (2) algorithms to find approximate solutions, and (3) algorithms that target only special cases for the problem for which better, easier, or exact solutions are feasible. mTSP is commonly solved by transformation to TSP. Exact solution algorithms mainly depend on relaxing some constraints of the problem or they work only on small problem sizes, e.g. Held-Karp algorithm using dynamic programming (Held and Karp, 1962) and Gavish and Srikanth (1986) algorithm using a branch-and-bound method. Approximate solutions give seemingly good solutions using heuristics. The literature has many heuristic algorithms such as nearest neighbor and Christofide heuristics. Other general optimization heuristics have been applied such as genetic algorithms, simulated annealing, Tabu search, and ant colony optimization. Refer to Johnson and McGeoch (1995) for more detailed description and comparison of these approaches to TSP. As for mTSP, some efforts have been done to find heuristic solutions using neural networks (NN). Wacholder et al. (1989) extended the Hopfield NN model for mTSP, but the model was too complex. Vakhutinsky and Golden (1994) and Modares et al. (1999) presented a self-organizing NN approach. Another direction is to adopt genetic algorithms (GA) to TSP applications such as hot rolling scheduling in Tang et al. (2000) and path planning in Yu et al. (2002). Sofge et al. (2002) gives a thorough comparison between various evolutionary algorithms in the scope of mTSP. Still mTSP differs from the problem of dividing seeking territories. It requires a graph representation and has more restricted objective to visit all the cities, while territory division is more probabilistic. In other words, our territory division problem is a more specific version of mTSP so that new methodology can be devised.

Another related problem to our work is the problem of map coverage for robotics. Choset (2001) thoroughly provides a well-structured survey of different approaches for both, single and multiple agents. Galceran and Carreras (2013) give another updated survey for the recent achievements made in the last decade. Since we are considering map coverage with multiple agents, we will only take into account relevant techniques like cellular decomposition as in Rekleitis et al. (2009) which uses the Boustrophedon coverage algorithm. Other pertinent category is to use spanning trees as a base for efficient coverage paths in multi-robot coverage. Agmon et al. (2006) exploited this idea by introducing a polynomial-time tree construction algorithm in order to improve the coverage time. However, they assume that the robots are initially distributed over the terrain and they build the spanning tree so that the robots do not pass by each other throughout the coverage plan. The problem of coverage for multi-robots is a little different from the territory division problem. In seeking tasks, it is commonly the case for the seekers to start at adjacent points and their seeking trajectories might intersect. Furthermore, it is not necessary to cover all the terrain in Hide-and-Seek; not only scanning the possible hiding places is sufficient, but scanning the good hiding places first is also preferable. These are the main differences that motivated our research. The previous discussion shows that although both mTSP and map coverage are related to our work, yet they are different in problem formulation and objectives. In Section 4, we propose a new learning model to solve the problem of territory division using a *hierarchical learning scheme.*

Between the limitation of the heuristic approach and the hardness of the exact solution, there is the learning approach that lies in between. By learning, it is possible for the system to adapt, through training, to different map topologies and different hiding probability distributions. As a result, the problem of being too-specific which some heuristic solutions mandate is eliminated in the learning approach. Additionally, designing a learning model is generally easier than deriving an approximation algorithm for such hard problems.

This work focuses on the learning approach to solve the territory division problem; by adopting the reinforcement learning paradigm which is widely used in Robotics. Reinforcement learning is different from both supervised and unsupervised learning in that agents learn through repeated interactions with the surrounding environment. Nevertheless, the RL system suffers from the curse of dimensionality when it targets hard problems with huge state spaces. This issue is discussed thoroughly in later sections. After the proposed learning model is introduced, it is enhanced to overcome the space explosion by the means of *state aggregation*. The next section gives an introduction to reinforcement learning showing the guidelines and the prerequisites for designing the learning model of standard RL system.

3 Reinforcement Learning

Reinforcement Learning is a machine learning paradigm in which the agent learns over time through repetitive trial-and-error interactions with the surrounding environment. Rewards are received by the agent depending on the agent's actions in the observed particular states of the environment. RL aims at learning an optimal decision-making strategy that gains the maximum total (discounted) reward received from the surrounding environment over the agent's lifetime.

The learning approach of RL differs from any supervised and unsupervised learning techniques. The learning is done through sequential experimentation with the environment rather than being trained using a static pre-prepared training set. That is why RL is more amenable to real-time applications with multi-agent systems. Nevertheless, this learning approach can be considerably harder for various reasons. First, it requires the agent to learn by letting his actions affect the environment, which requires the knowledge of a complete (or at least partial) model for the environment, the agent, and the actions' effects. Second, even the learning feedback itself can be complicated. The reward gained by applying some policy depends on the system state, the applied action, and the dynamics of the environment. Besides, unlike the labeled training set in supervised learning, the feedback is only immediate for the current action, whilst the learning process tries to maximize the long-term cumulative total reward that will be gained in the future. Currently, several RL algorithms have been introduced and have shown great success both on theory and applications such as *Q*-learning (Watkins and Dayan, 1992), TD(λ) (Sutton 1988), Sarsa-Learning (Singh et al., 2000). Kaelbling et al. (1996) provide a fine survey of the research done in RL and its related topics.

Most RL algorithms are based on the Markov Decision Process (MDP) model in which the agent needs to make a decision to choose an action *a* depending only on the current state *s* (irrespective of the historical state-action sequences) resulting in a new state *s'*. A cumulative reward function is used to quantify the quality of the temporal sequence of actions. There are two main learning schemes of RL. One is model-free learning in which the agent learns the optimal policy without knowing an explicit model of the environment, in other words without knowing the parameters of the Markov process (the transition and reward functions). The other scheme is model-based learning in which the agent learns the model of the environment (the Markov process) whilst searching for an optimal strategy. Sutton and Barto (1998) in their book give a detailed introduction into the concept of Reinforcement Learning and its different methods. They describe three fundamental classes of methods to solve the RL problem: dynamic programming, Monte Carlo method, and temporal-difference (TD) learning. Each of which has its pros and cons. For example, dynamic programming methods are based on rigorous mathematical formulation, but require a full model for the environment. Monte Carlo methods are much simpler as they do not require a complete knowledge of the environment; however, they are well suited only for episodic tasks in which the updates from learning increment episode-by-episode rather than action-by-action. Finally, temporal-difference methods are a mix of the two previous approaches of dynamic programming and Monte Carlo. TD learning is incremental through actions and does not require a full model of the environment's dynamics, however, it is more complex to analyze. *Q*-learning is one learning algorithm based on the TD learning paradigm. In our work, we investigate *Q*-learning in more details and use it in the hierarchical learning task of Hide-and-Seek.

3.1 *Q*-Learning

Watkins and Dayan (1992) introduce the *Q*-learning algorithm. The agent’s task is to learn an optimal policy $\pi^* : S \rightarrow A$, that tells the agent which action $a \in A$ to take given the current state $s \in S$ with the goal of maximizing a cumulative discounted reward. Given a policy π and a state s_t at time step *t*, the valuation of s_t under π (assuming infinite horizon) can be calculated as follows:

$$V^{\pi}(s_t) = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots \tag{3.1}$$

Where $0 < \gamma < 1$ is the *discount factor* and r_{t+1} is the immediate reward at time t. Thus by calculating $V^*(s)$, the optimal value function, an optimal action policy π^* is readily available. Unfortunately, in many practical problems the reward function (and generally the environment model) is unknown which means Eq. (3.1) cannot be evaluated, therefore, the need for learning. Let $Q(s,a)$ be the maximum discounted cumulative reward achieved by taking action a from state s and then acting optimally.

$$Q(s,a) = R(s,a) + \gamma V^*(s') \tag{3.2}$$

where $R(s,a)$ is the immediate reward function, and s' is the next state reached from state s after taking action a (this formula assumes deterministic environment, but can be easily extended to the probabilistic case). Notice that $V^*(s) = \max_a Q(s,a)$ which allows us to rewrite Eq. (3.1) in a recursive form as follows:

$$Q(s,a) = R(s,a) + \gamma \max_{a'} Q(s',a') \tag{3.3}$$

The Q-learning algorithm constructs a table for its estimated Q values and observes the $\langle s,a,r,s' \rangle$ experience tuple in each iteration (interaction with the environment). After visiting every state-action pair infinitely often, the constructed Q table is guaranteed to converge to its optimal value (Leng et al., 2007). Fig. 3.1 shows the design of a Q-learning system and the interaction between the agent and the environment.

A standard Q-learning system consists of three main components. First, the Environment block which represents the world that the agent observes and interacts with, it provides the agent with the current state s and the immediate reward $R(s,a)$ resulting from the action a just executed by the agent. The other two components belong to the agent. The Q-learning unit which is the core of the Q-learning algorithm; it contains the Q-table $\{Q(s,a)\}$ and the updating rule. Finally, the Action Strategy unit is responsible for deciding which action a to take and how to handle the tradeoff between exploration and exploitation.

The convergence theorem of Q-learning implies that learning the optimal Q function can be achieved even by acting with a complete random action sequences, as long as every state-action pair is visited infinitely often. That is why Q-learning is considered one of the simplest and most popular model-free learning techniques. However, convergence is only guaranteed assuming exact discrete representation of the state-action pairs, which is not feasible in real world applications. In addition due to the inefficient use of experience, the large-scale state-action space which is a typical case with real life problems makes Q-learning suffer from the curse of dimensionality. Consequently, the computational time and memory space requirements are prohibitive for most practical applications. These issues are tackled by the techniques of generalization, which is briefly discussed in the next section.

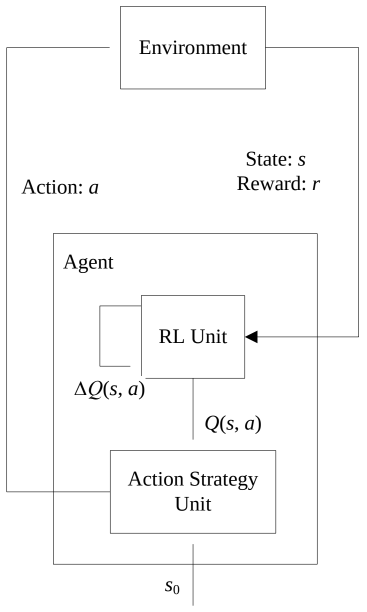


Fig. 3.1. The design of Q-learning system with agent-environment interactions.

3.2 Generalization techniques

Unfortunately, almost all the available theories of convergence of RL algorithms assume the use of a lookup table representation for the value function. This highly increases the computation and space complexity of RL techniques. So introducing more compact representation is crucial for many practical applications, and that is what the generalization techniques do (Kaelbling et al., 1996). Some of these generalization techniques use *function approximation* such as artificial neural networks (ANN) (Tang et al., 2007), adaptive resolution models, hierarchical learning, and state aggregation. S*tate aggregation* is the generalization technique that we applied in our work.

Generally, the goal of the generalization techniques is to reduce the state space such that the learning process becomes computationally feasible both in time and storage requirements, whilst keeping as much as possible the convergence guarantees. One generalization approach is the state aggregation (Singh and Jaakkola, 1995). Most often, the state space is smooth, i.e. the values of adjacent states are equal or slightly different. This property is the cornerstone of state aggregation techniques. The basic idea of state aggregation is to combine similar states of the large state space to form one aggregate state so that the agent learns over a reduced abstract Markov decision process (Marthi 2007; Bertsekas and Castanon, 1989).

This introduces two issues that need to be considered: (1) how to determine the similarity between two states i.e. how to construct the aggregate states and (2) how to learn over these aggregate states where a new action space, a new reward function, as well as a new state space ought to be defined properly. In Section 5, we describe how the state aggregation concept can be embodied into the *Q*-learning process.

4 Reinforcement Learning in Territory Division

As discussed earlier, in a game with multiple cooperative seekers a good seeking strategy is to properly divide the territories among the seekers in order to increase the seeking efficiency and thus decrease the seeking time. To achieve a good territory division strategy, agents should learn how to build these strategies by themselves. In this section, a new learning model with RL is presented to solve the multi-seeker territory division problem using a hierarchical learning scheme. The following subsection explains the overall idea of the model and its hierarchical structure. This is followed by a detailed description of the learning model. Finally, the proposed model is examined through some experimentation where the performance is also discussed.

4.1 Sequential decision tasks

Singh (1992a, 1992b) introduces an elegant efficient learning model for a task consisting of a sequence of multiple subtasks for a single agent. It considers a learning agent that deals with a finite-state environment aiming at performing a sequence of tasks in a discrete-time manner. Each task requires the agent to take a sequence of actions to bring it to the next desired target. The territory division problem in a Hide-and-Seek game can be formulated similarly as follows:

1. Given a discrete finite environment, a grid space $E \subset \mathbb{Z}^2$. Each point in the space can be either obstacle or free.
2. Given a set of seeking agents, each of which is a moving point in E and at any arbitrary time $t \in [0,T]$ has a location $loc_t \in E$. Agents cannot possess a location with an obstacle in it.
3. At any arbitrary time $t \in [0,T)$, each agent can move to a neighboring point, if possible, by taking an action $a_t \in \{N, S, E, W\}$.
4. Given a set of points $H \subseteq E$ which are possible hiding places with likelihood function $P_{hiding} : E \to [0,1]$, such that $\forall h \in H : P_{hiding}(h) > 0$. Each of which acts as a target for any agent to visit.
5. At any arbitrary time, define a set of visited points $V_t \subseteq H$ such that $V_0 = \varnothing$ and $V_T = H$. An agent visits a target point at time t when its location $loc_t \in H$ and $loc_t \notin V_t$, then $loc_t \in V_{t+1}$.

6. The problem is to find the seeking trajectory for each agent that optimize the objective function: maximize $\sum_t \sum_{agents} \gamma^t P_{hiding}(loc_t)$, such that $0 < \gamma < 1$.

A good seeking trajectory for each agent should divide the search space into territories for the sake of the objective function. Our model divides the learning process in the territory division problem into two-level scheme; there are two kinds of tasks: "elemental" tasks in the lower level and "composite" task in the higher level. Assume a seeker is assigned an arbitrary part of the environment to be his seeking territory, and then the seeker has to perform a sequence of elemental tasks. Each *elemental task* corresponds to visiting one possible hiding place. The job is completed after doing a sequence of such tasks in order for the seeker to visit all the possible hiding places in his assigned territory. Nonetheless, the crucial part is the territory allocation, which is to determine a set of hiding places that would be assigned for each seeker, and to find the best order of visiting, this task is encapsulated as a *composite task* that consists of a sequence of elemental tasks.

An elemental task can be independently solved by straightforward techniques. RL algorithms such as *Q*-learning or Sutton's TD(λ) can be used to solve each elemental task independently as a simple path planning problem. Our previous work in Gunady and Gomaa (2012) gives an example of learning for path planning. This independence makes it easy in some special cases, e.g. in a small and deterministic problem space, to directly apply graph-theoretic algorithms such as Dijkstra's shortest path algorithm (Dijkstra 1959), if applicable. Regardless of the technique used to solve the elemental tasks, having these tasks solved means that the seeker knows the best strategy to visit each hiding place in the environment from its current position. As a result the territory division problem can be reduced to the problem of learning the best order of visiting all the hiding places in the minimum expected total time. In our work, we focus on solving this composite task with the assumption that the elemental tasks are already solved in earlier process as discussed. The hierarchical learning scheme is illustrated in Fig. 4.1. A single basic block: *Q*-learning Hiding place #*i* is responsible for learning the best strategy to reach the hiding place number *i* from any current location. Notice that if the elemental tasks can be solved by shortest path techniques, these learning blocks can be replaced with other computational blocks of shortest path techniques.

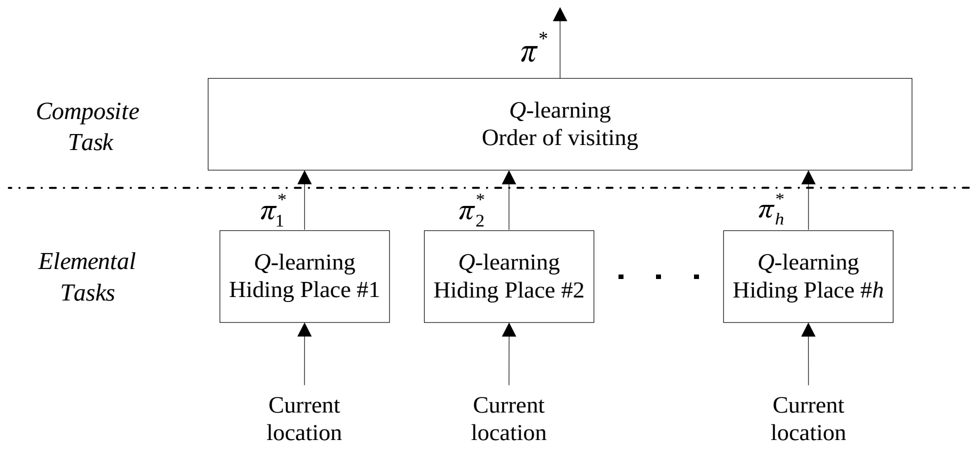


Fig. 4.1. Hierarchical structure of a sequential task learning for *h* hiding places.

4.2 The learning model

This subsection presents the details of the hierarchical learning scheme explained in the previous section. The proposed model solves the composite task at the higher level. The first subsection discusses the case of single-agent and the second subsection generalizes to the multi-agent configuration. Although the single-agent case may seem trivial when speaking about the territory division problem, it is just a hypothetical case used for illustration. Considering this case helps in building the learning model for a simplified setting first, and later the model can be generalized to the multiple seekers case.

4.2.1 Single agent learning

In this subsection, we develop a model for the learning blocks in the higher level: *learning the task of territory division as a sequence of elemental tasks*. Consider a simple Hide-and-Seek game with a single seeker. Assume that the seeker knows the hiding probability distribution used by the hider through prior knowledge or through early stages of learning. In addition, the seeker has already learned how to find the shortest path from any location to each of the possible hiding places, i.e. the elemental tasks. Now, the seeker's task is to learn the best visiting order for these possible hiding places, which minimizes the expected time to find the hider.

As for the state space, each state consists of two parts: the set of already visited hiding places V (an unordered set; as the actual order in the history does not affect the choice of the future actions) and the current location *loc* of the seeker that represents the location of the last visited hiding place. Let H denote the set of all possible hiding places in the environment, a set of size h. And hence a state is defined as $s \equiv \langle V, loc \rangle$. Now given state s_t at time t, we define the action to be the next hiding place to be visited by the seeker. The transition function δ given the current state s and action a is defined as follows:

$$\delta(s,a) = \langle V + a, loc' \rangle = s' \tag{4.1}$$

where $loc' = a$ the action taken by the seeker. The reward function is defined as a function of the hiding probability for the resulting location, where w is a reward constant, set to 10, as follows:

$$R(s,a) = w\ P_{hiding}(loc') \tag{4.2}$$

Where for any location *loc*, $P_{hiding}(loc)$ is the probability that the hider would use the location *loc* as a possible hiding place. It should be noted that the above action definition leads to unequal time steps for executing different actions because the length of the actual actions taken to reach the next hiding place is different depending on the current location of the seeker and the location of the targeted hiding place. That requires the standard updating rule of Q-learning in Eq. (3.3) to be revised. Consider a learning process at time t; it is required to perform the transition $\delta(s_t, a_t) \xrightarrow{\tau} s'_{t+\tau}$ that takes transition time τ for execution. Starting from Eq. (3.1), the Q value formula can be written as follows:

$$V^{\pi}(s_t) = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots$$

This can be rewritten as:

$$V^{\pi}(s_t) = (r_{t+1} + \gamma r_{t+2} + \ldots + \gamma^{\tau-1} r_{t+\tau}) + \gamma^{\tau} r_{t+\tau+1} + \ldots$$

$$Q(s_t, a_t) = \left[\sum_{k=0}^{\tau-1} \gamma^k r_{t+i+1} \right] + \gamma^{\tau} V^{*}(s'_{t+\tau}) \tag{4.3}$$

From the definition of the reward function all intermediate transitions have zero reward. Therefore, $r_{t+i+1} = 0$, for $i = 0, 1, \ldots, \tau - 2$. Thus, the new updating rule will be as follows:

$$Q(s,a) = \gamma^{\tau-1} R(s,a) + \gamma^{\tau} \max_{a'} Q(s', a') \tag{4.4}$$

The impact of the discount factor $\gamma^{\tau-1}$ reflects the effect of the distance to the hiding places which is absent from the reward function. The new updating rule successfully preserves the total history length through the recursive formula, which is crucial for convergence to the optimal strategy. There is one last thing to be clarified before the model gets more complicated with multi-agent learning. It should be noticed that this model deals with the learning task in the higher level only. So when mentioning the word *action* for example, it deals with the abstract action that the agent takes in order to make a transition from an abstract state to another (corresponding to visiting possible hiding places). This is absolutely different from the sequence of physical low-level actions taken by the agent to realize that transition. The latter sort of actions is dealt with in elemental tasks in the lower level. To avoid any confusion, we would use the term *ground actions* when referring to the physical low-level actions, while using the term *abstract actions* or just *actions* when referring to actions in the higher level learning process. In addition, we say an agent is at an *intermediate state* when it is still transiting from a state to another by taking an abstract action.

4.2.2 Multi-agent cooperative learning

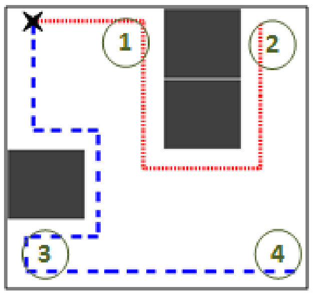
We can extend the model discussed in the previous subsection for the case of *k* agents. First of all, a global view for the whole system state must be preserved, which can be realized by providing full communication capabilities among the seekers in the learning phase. This capability is required only within the learning phase, while it is not required at application phase. The state definition would be $s \equiv \langle V, Loc \rangle$ where *Loc* is a vector of the current locations of all *k* seekers, e.g. $Loc \equiv \langle loc(1), loc(2), \ldots, loc(k) \rangle$ where at least one of *loc*(*i*) is a hiding place (except for the initial state). The action definition would be $a \equiv \langle a(1), a(2), \ldots, a(k) \rangle$, such that at time *t* an action $a_t(i)$ of seeker *i* is given by Eq. (4.5). Table 1 illustrates a pseudo-code of the action selection procedure at each time step.

$$a_t(i) = \begin{cases} p \in H - V_t \text{ such that } p \neq a_t(j),\ j \neq i & \text{if } t = 0 \text{ or seeker } i \in G_t \\ a_{t-1}(i) & \text{otherwise} \end{cases} \tag{4.5}$$

where G_t is a set of seekers who reach their targets at time *t*. Action *p* is an unvisited hiding place which is not currently selected by any other seeker as the next target. The $a_{t-1}(i)$ clause means that the agent *i* is in an intermediate state. A seeker in an intermediate state is performing a sequence of required ground actions in order to transfer from the previous hiding place to the next one. The essence of that clause appears only in the case of multiple seekers; as the time steps (required to execute abstract actions) are not equal anymore. The time required to perform $a_t(i)$ may not be equal to the time required by $a_t(j);\ i \neq j$. That means one seeker will reach his next state while the others are still in intermediate states. Note that Eq. (4.5) is described in terms of the time step of the ground state. When G_t is not empty, the state is an abstract state; hence we can write the same non-empty set as G_s in the abstract state space. Fig. 4.2 shows an example of action interpretation with time steps for two seekers in a 4×4 grid map with discrete ground action space {N, S, E, W}. Numbered circles represent four possible hiding places. The two seekers start from the map origin (1, 1) in the top-left corner. The dashed lines represent the seeking trajectories of the two seekers.

Table 1. Pseudo-code of the action selection procedure at each time step.

```
Given time $t$, current state $s_t$ , previous action $a_{t-1}$:
Define $T$ as the hiding places which are currently the target of any seeker; i.e. $T = a_{t-1}$.
FOR each seeker $i$
  IF $t=0$ or seeker $i$ reaches its target
    Get set $\mathcal{H}$ of all unvisited hiding places.
    IF $|\mathcal{H}| \geq$ number of seekers
      Remove from $\mathcal{H}$ any hiding place which is currently the target of some other seeker;
        i.e. $\mathcal{H} = \mathcal{H} - T$.
    End IF
    Choose a hiding place $p$ at random from $\mathcal{H}$ to be the next action $a_t(i)$.
    $T = T - a_t(i)$.
  ELSE (seeker is in intermediate state)
    Use previous action $a_{t-1}(i)$ to be the next action $a_t(i)$.
  END IF
END FOR
```



$t = 0$	$Loc_t =$ <(1,1),(1,1)>	$a_t =$ <1, 3>
$t = 1$	$Loc_t =$ <(1,2),(2,1)>	$a_t =$ <2, 3>
$t = 5$	$Loc_t =$ <(3,4),(4,1)>	$a_t =$ <2, 4>
$t = 7$	$Loc_t =$ <(1,4),(4,3)>	$a_t =$ <4, 4>
$t = 8$	$Loc_t =$ <(2,4),(4,4)>	Game terminates!...

Fig. 4.2. An example of location-action interpretation for two seekers in a 4×4 map.

The reward function for the multi-agent case is a generalized form of the single-agent case; the total immediate reward is the summation of the individual rewards obtained by each seeker's action as in Eq. (4.6). Each individual reward is a function of a seeker's individual state and action, and calculated as in section 4.2.1.

$$R_{joint}(s,a) = \sum_{\text{seeker } i \in G_{s'}} w \, P_{hiding}(loc'(i)) \tag{4.6}$$

Similarly, the Q updating rule is reformulated to include the joint state s, action a, reward function $R_{joint}(s,a)$, and transition time τ from s to s' as follows:

$$Q(s,a) = \gamma^{\tau-1} R_{joint}(s,a) + \gamma^{\tau} \max_{a'} Q(s',a') \tag{4.7}$$

4.3 Experiments

This subsection shows the experiments used to validate our model. A set of 2D grid maps are used as a test-bed for the proposed learning approach. Once again, a seeker has ground action space {N, S, E, W} used in the lower level of learning. Results are to be shown, criticized, and compared with the expected optimal solution.

Fig. 4.3 shows a set of different maps used for experimentation with different sizes, topologies, and different hiding probability distributions. The maps are of small toy size for proof of concept of the learning process and to facilitate studying its effectiveness. The X on each map marks the starting position of the seekers, while black cells represent blocks that seekers cannot go through. Cells with black circle or triangle mark are possible hiding places with different hiding probabilities, for instance a triangle-cell has hiding probability three times higher than a circle-cell. Experiments are done for two seekers on each map. Fig. 4.4 shows the resulting seeking trajectories of the two seekers to which the learning procedure converges. The dashed lines represent the seeking trajectories while the solid line is an intersection between the two trajectories. It should be noted that the best seeking strategy does not require the seekers to visit all the cells, but rather only those related to the possible hiding places. Furthermore, it is not necessarily preferred to visit the nearest hiding place first, because it may have low hiding probability, which might increase the expected time to find the hider. The results of map (e) illustrate an example of such case: both seekers prefer to visit far but highly probable hiding places, and then backtrack to hiding places with lower probability.

Over time, the maximum Q value of the initial state with all its possible actions gives an indication for the progress of the learning process; as it converges to the optimal value gained by applying the optimal policy. For verification purposes, each test case is solved by doing an exhaustive search; which tries out all the possible solutions and chooses the seeking sequence that gives the minimum expected time for finding the hider. The comparison between this solution and the solution derived by learning showed that the proposed system converges towards the optimal solution for all cases listed. Fig. 4.5 shows the convergence progress towards the optimal solutions. The maximum Q values of the initial state learnt so far are compared with the actual cumulative discounted reward gained by the optimal solutions. As for the action selection method, actions are chosen probabilistically based on the estimated Q value of the state-action pairs.

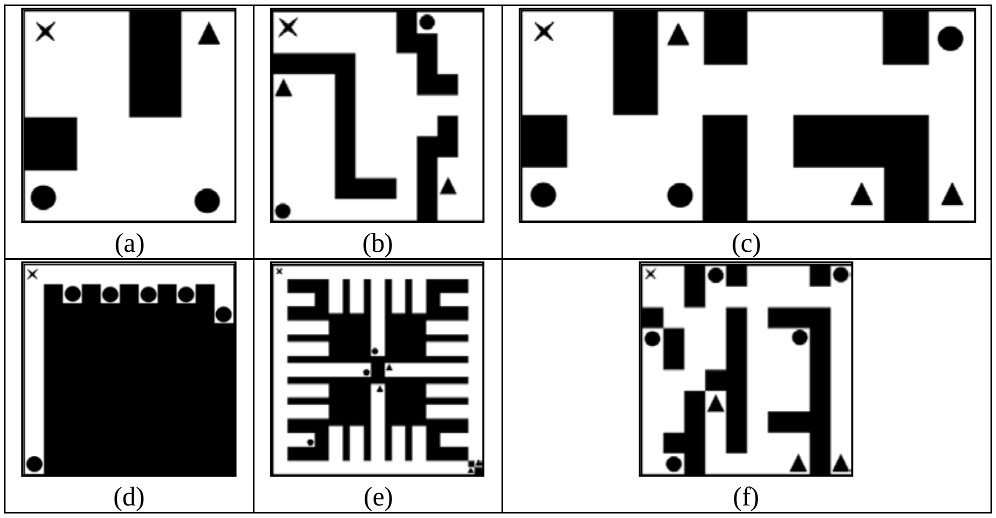


Fig. 4.3. Different maps used for the experiments.

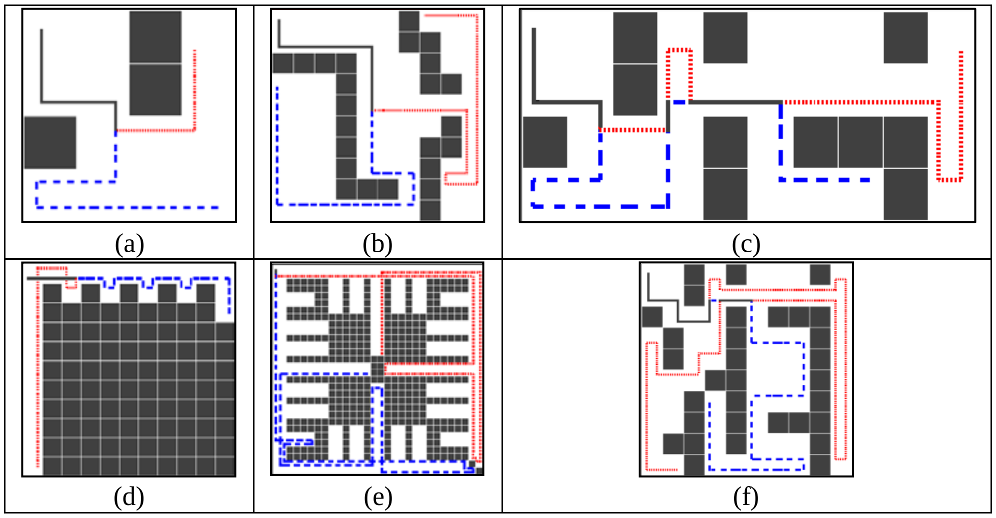


Fig. 4.4. Resulting seeking trajectories for experiments of two seekers.

As discussed in the introduction of this section, the problem of multi-agent territory division is similar to the mTSP which is classified as *NP-hard*. So it is expected for the learning approach to suffer from similar problems. Nevertheless, the learning approach has an advantage over a brute-force searching algorithm. First, it does not require a rigorous understanding for the whole problem details; our proposed learning model can be applied to different variations of task decomposition with few adjustments. Second and more important, the learning process can give a good sub-optimal solution within a reasonable time. This sub-optimal solution gets closer to the optimal solution by allowing more learning time; this is not the case with a non-learning searching algorithm. This can be observed from the results in Fig. 4.5. Increasing the problem size in terms of increasing the number of hiding places and the map size, the number of territory division learning episodes, or simply episodes required till convergence highly increases, where an episode is a variant game in which seekers' objective is to visit all the possible hiding places. For example, map (a) of size 4×4 has only 3 hiding places and it required only 5 episodes till convergence. Whereas map (f) of size 10×10 has 8 hiding places and it required 30,000 episodes, whilst a good sub-optimal solution which is very close to the optimal one is found within 1,500 episodes.

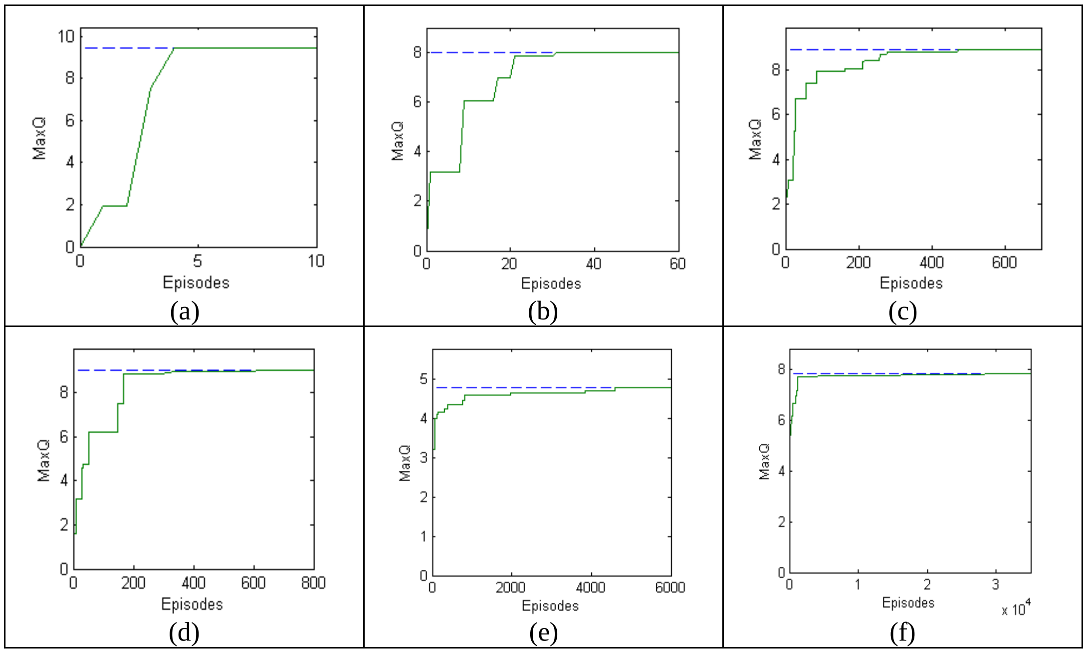


Fig. 4.5. The convergence progress over time, γ = 0.99.

4.4 Complexity analysis of state space

In order to evaluate the experimental results, it is necessary to perform some complexity analysis to understand the behavior of the proposed learning model. In RL, the main factor affecting the learning process, in terms of convergence time and space requirements, is *the size of state-action space*. As discussed in the introductory section, convergence is based on visiting every possible state-action pair infinitely often. In other words, the bigger the state-action space, the more complexity the solution requires. In our learning model, an abstract action is defined to be one of the unvisited hiding places. Thus the action space size is trivially a linear function of the number of hiding places *h*. In the case of multi-seeker: given *k* seekers, the size of the joint abstract action space is $O(h^k)$.

As in section 4.2, a state $s = \langle V, Loc \rangle$ is defined as a pair of two components: *V* which represents the set of already visited hiding places, and a vector *Loc* of the current locations of the seekers. For simplicity, an analysis is done for the case of two-seekers and then it is generalized to *k* seekers. Recall that a state transition occurs only when at least one of the seekers reaches a possible hiding place. In other words, a new state is defined when: (1) both the seekers are at possible hiding places or (2) one of them is at a possible hiding place while the other is in an intermediate state (on the path from an already visited hiding place to the next hiding place). Starting with the first and simpler case: when both the seekers simultaneously reach two possible hiding places, this gives the number N_1 of possible states:

$$N_1 = \sum_{i=1}^{h} \binom{h}{i} i(i-1) \tag{4.8}$$

where if there are $i \le h$ already visited hiding places, there would be $\binom{h}{i}$ possibilities – noting that the order is not important, while there would be *i* possible locations for one seeker and $(i-1)$ other possible locations for the other seeker. That sums up to the total N_1 states.

Now let's consider the second case: one seeker is at a possible hiding place, while the other is in an intermediate state. The factors that determine an intermediate state are: the last visited hiding place from which the seeker is moving, the next hiding place to be visited by the seeker executing his current action, and how much time has passed since the beginning of this intermediate transition. The last factor is bounded by the maximum distance between any two hiding places. This maximum distance is bounded by the map size. Hence in a rectangular grid map of size n^2, the last factor is bounded by n^2. This gives an upper bound of the number N_2 of possible states for the second case as follows:

$$N_2 \le \sum_{i=1}^{h} \binom{h}{i} i \ [(i-1)(h-i)n^2] \tag{4.9}$$

Summing up the two cases (plus 1 for initial state) bounds the total number of states as follows:

$$|S| \le 1 + \sum_{i=1}^{h} \binom{h}{i} i\,(i-1)[1+(h-i)n^2] \tag{4.10}$$

This formula gives an upper bound of the state space size for two seekers case, which is asymptotically $O(2^h\ n^2)$. That can be generalized for more than two seekers. Following the same line of reasoning, a generalized formula can be derived for a general case of k seekers as follows:

$$|S| \le 1 + \sum_{i=1}^{h} \binom{h}{i} \sum_{j=1}^{min(k,i)} \left[\prod_{l=1}^{j} (i-l+1) \right] \left[\prod_{l=1}^{k-j} (i-j-l+1)(h-i-l+1)n^2 \right] \tag{4.11}$$

which means the state space size will be asymptotically $O(2^h\ n^k)$ for general k seekers case. This analysis shows how the complexity grows exponentially with the number of hiding places. The impact of the map size is relatively small compared with the number of hiding places; it is typical to have number of agents much less than the number of tasks. However, by increasing the number of seekers, that impact becomes more involved, e.g. if $k = h$, then the n^k factor becomes dominant.

4.5 Section summary

In this section, the problem of learning multi-agent territory division in Hide-and-Seek games is discussed. The target is not to perform a full coverage of the terrain, but scanning only the possible hiding places so that the total expected time for finding the hider is minimized. That requires finding the optimal sequence of visiting the hiding places according to the hiding probability distribution. We present a new learning approach to solve this problem that is based on reinforcement learning with an illustration of how the learning system is built in a hierarchical structure. The elemental tasks of path planning for each hiding place is learnt first in the lower layer, then the composite task of finding the optimal visiting sequence is learnt in the higher layer. The full module has been described including the state definition, the action definition, and the reward function for both single and multiple seekers. A revised version of the standard updating rule of Q-learning is introduced.

This new equation is essential to deal with the problem of unequal time steps for different agents' actions. The proposed approach is examined on a set of different maps, in which the learning process converged to the complete optimal solution. The impact of the problem size and the complexity of the system are also studied and formulation is derived. The results show that the proposed learning approach has advantages over the heuristic and algorithmic techniques in that: it is easier to design, generic and can be applied to different variations of task decomposition, and gives a good near-optimal solution within a reasonable time for problems of large state-action space. However, because the problem itself is inherently hard, the learning model suffers from the time and space explosion with respect to the problem size. In the next section, we solve this problem by applying the state aggregation concept in our learning model.

5 Learning with State Aggregation

Through the previous experiments, it was shown that the learning problem of a multi-agent territory division suffers from the curse of dimensionality in the state space. Usually, space explosion problem in reinforcement learning is tackled by generalization techniques such as ANN, hierarchical learning, and state aggregation. We exploit *state aggregation* in our learning model. The idea is to group *similar* states together into one aggregate state, and then learn over the aggregate level, which is highly reduced in size.

Generally, generalization techniques such as state aggregation are approximation techniques. As a result, it is not guaranteed for the system to give the exact optimal solution, although a near optimal solution is achieved within reasonable time and space requirements. In this section, we first illustrate how the state aggregation technique can be embodied into the RL process. Afterwards we upgrade the idea to be used in the actual problem of territory division learning. We present how the learning model in the previous section can be extended to include the aggregation layer. Finally, our experiments show significant improvements both in time and space.

5.1 State aggregation in *Q*-learning

As mentioned in the background section 3.2, there are two issues to be considered while designing the State-Aggregation *Q*-learning (SAQL) system. First is how to determine the similarity between two states, i.e. how to construct the aggregate states. Second is how to learn over these aggregate states for which new action space, new reward function, as well as new state space needs to be defined.

In a standard RL system, the problem is modeled as an MDP with defined state and action spaces. On the other hand, when applying the state aggregation technique, new state and action spaces are defined, referred to as the aggregate spaces. The aggregate state space and the aggregate action space are hypothetical spaces that are formed by aggregating the original spaces (the reduced MDP). Let $\hat{S}$ denote the aggregate state space, $\hat{A}$ the aggregate action space, and $\hat{R}$ the new reward function. The aggregate state space can be constructed by using some clustering technique applied over the original states. Then the action space will be customized to form a new aggregate action space. For example, we can group neighboring states that have equal reward values into one aggregate state and the value of this state generalizes its constituent states. Learning over the hypothetical aggregate space is essentially similar to that of learning over original space. The standard *Q*-learning system shown in Fig. 3.1 is modified as shown in Fig. 5.1 to include the *state aggregation unit*, which acts as a wrapper of the real environment from the agent's perspective. Each interconnection between two blocks corresponds to a major step in the SAQL algorithm (Gunady and Gomaa, 2012).

The architecture of the SAQL system is similar to the standard *Q*-learning system. The main difference is the embedding of the State Aggregation unit within the interconnections between the agent and the environment. During the learning process, all variables will be transformed from the original space to the aggregate space. The only step in which the agent has to deal with the real environment is when it tries to execute the selected aggregate action, from step 3 in Fig. 5.1, in the real environment. In spite of the fact that the agent learns over the aggregate space, that aggregate space remains hypothetical whilst actual actions have to be realized in order to continue exploring the problem space. That introduces step 4 in Fig. 5.1 in which the agent has to take a sequence of actions $a_1 a_2 \ldots a_n$, in order to realize the aggregate action $\hat{a}$. The results discussed in Gunady and Gomaa (2012) show how our SAQL system is effective in reducing the state space size and improving the convergence rate of *Q*-learning even with simple grouping techniques. That proves how state aggregation is promising in RL generalization to solve the problem of state-action space explosion.

5.2 Aggregation in territory division learning

After studying the basic concept of using state aggregation in *Q*-learning system, we can enhance our proposed learning model presented in Section 4 with the technique of state aggregation. To achieve any reduction in the state space, the state definition terms should be considered. As a quick review, our learning model described in section 4.2 defines learning state as $s = \langle V, Loc \rangle$, where V represents the set of already visited hiding places, and Loc is a vector of the current locations of the seekers. Therefore, if the objective is to achieve a reduction in the state space, then a reduction should be applied in either of the two terms, or both. In that sense, we propose a state aggregation scheme that is applied in two directions: topological aggregation and hiding aggregation. In topological aggregation, the idea is to divide the map into regions according to the distribution of the obstacles, i.e. the map topology. This direction intends to achieve reduction in the Loc term in the state definition. By intuition it would be a good seeking plan for a seeker to scan close hiding places together before moving to other relatively far hiding places. Therefore, the idea of hiding aggregation is to group close hiding places together and then deals with the group as the target of aggregate action. Hence, a reduction in the number of hiding places V will be achieved.

5.2.1 System Structure

In our hierarchical model, state aggregation scheme can be applied on each learning layer. Topological aggregation appears in the lower-level layer so that instead of dealing with the current location as an input, it will take the current region to which the current location belongs. However, the hiding aggregation scheme appears in the higher-level layer. Fig. 5.2 shows the modified version of the learning model in Fig. 4.1. It should be noted that by introducing the state aggregation method, as in any other generalization technique, the solution is no longer exact, but approximated. Each aggregation layer adds approximation errors to the final solution, so it is essential to be careful with the extent of aggregation used to avoid bad global approximation, which may produce a bad solution far from the optimal one.

5.2.2 Topological Aggregation

In the topological aggregation, the target is to divide the map into geographic regions according to the map topology, i.e. the regions formed by the obstacles in the map. The concept is that: the exact current location of the seeker does not significantly affect his next action decision. For instance, if a seeker in location *loc* and took the decision to visit a near hiding place rather than another relatively far one, it would be most likely the same decision if its location is instead a unit distant from *loc*, or two, and so on. That is how we intend to exploit the topological aggregation concept to create map regions. The seeker should discover the regions that group close locations, which do not have a significant impact on his action decision. In the learning model, we replace the *loc* term in the state definition with the region to which the physical location *loc* belongs.

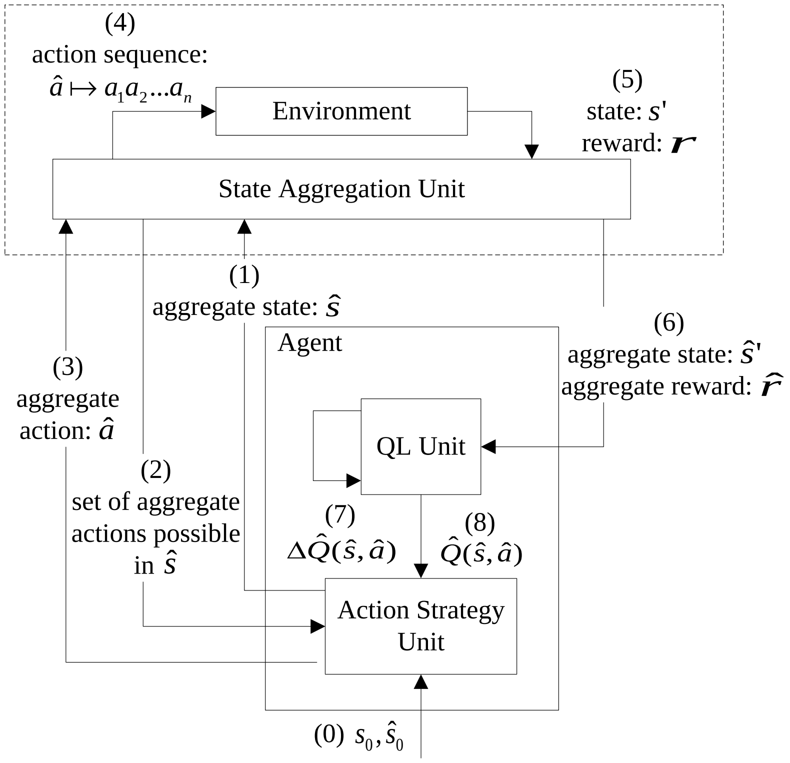


Fig. 5.1. *Q*-learning system architecture with *state aggregation*

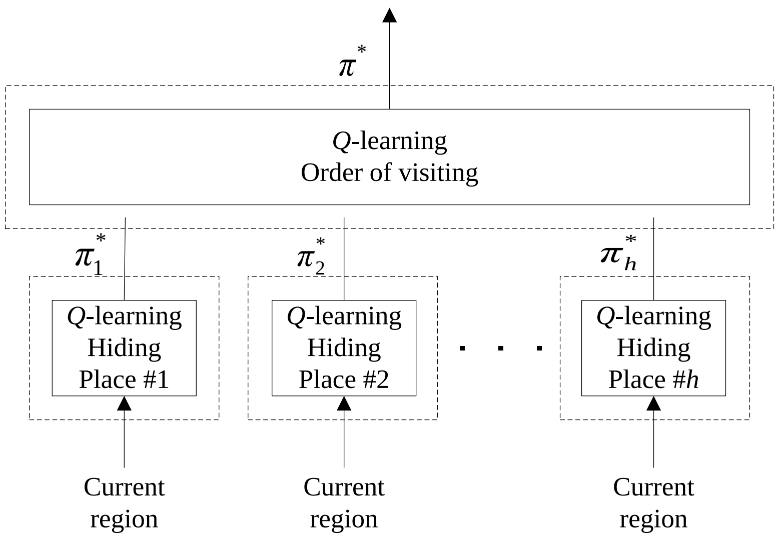


Fig. 5.2. Modified hierarchical structure of the learning model with state aggregation.

The aggregates or regions are globally constructed within the early stage of discovering the map topology. While the seekers discover new areas from the map, they construct arbitrarily shaped regions according to two aggregation parameters:

- Region Size (*maxSize*): The maximum number of cells that belong to a single region. This parameter is crucial for constraining the region size in order to avoid dominant regions, e.g. the whole map can be grouped into a single region!
- Region Diameter (*Dia*): The maximum distance between any two cells belonging to the same region. This parameter is important to prevent skewed regions, which give bad approximation.

Then each region is represented by one cell inside it. This representative cell plays an important role throughout the learning process. We use a simple greedy algorithm to construct the regions and use the geometric median as a sound heuristic for calculating the regions representatives. Our intuitive choice of geometric median is to minimize the average approximation error (distance between the region's representative and a cell which belongs to that region). Fig. 5.3 shows an example of the resulting regions constructed by the seekers with *maxSize* = 25, *Dia* = 15. A cell marked with dot is the representative of its corresponding region. The state space is hence reduced. Without topological aggregation, the *Loc* portion of the state definition takes up to 657 possible values for each seeker. Fortunately, with topological aggregation, it takes up to 43 possible regions for each seeker.

5.2.3 Learning with Topological Aggregation

With the state definition $s = \langle V, Loc \rangle$ in section 4.2, there can be multiple situations with the same visited hiding places *V* and the same action vector *a* but with slightly different vectors of the seekers' locations *Loc* due to different game scenarios. Consequently, each situation will be represented by a separate state. By embedding this level of aggregation in the learning process, we aim at reducing the state space by mapping all these situations into a single state. One solution is just to apply this mapping by replacing the vector of the current locations *Loc* with a vector of the representatives of the corresponding regions $\hat{s} \equiv \langle V, Rep \rangle$. In addition, use the new state definition when dealing with the *Q*-table while the rest of the system remains as it is. Although this solution is simple, it suffers from a fatal problem. Consider the two situations in Fig. 5.4. The two situations have the same state-action pairs. However, they have different transition states and hence different reward values. Such situations may lead to oscillations in the estimated *Q* values. Thus, we propose another solution.

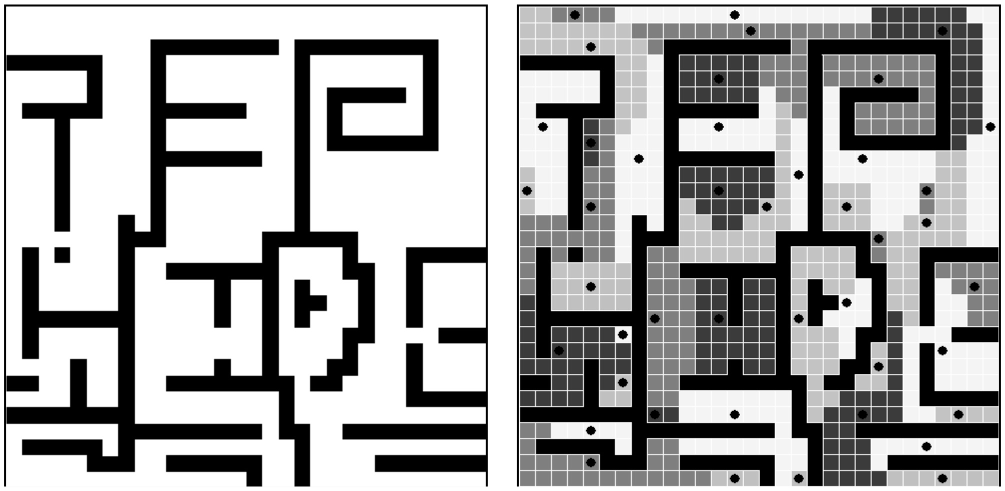


Fig. 5.3. An example of topological aggregation. *maxSize* = 25, *Dia* = 15. Within each region, a cell with a dot is the region representative (*geometric median*).

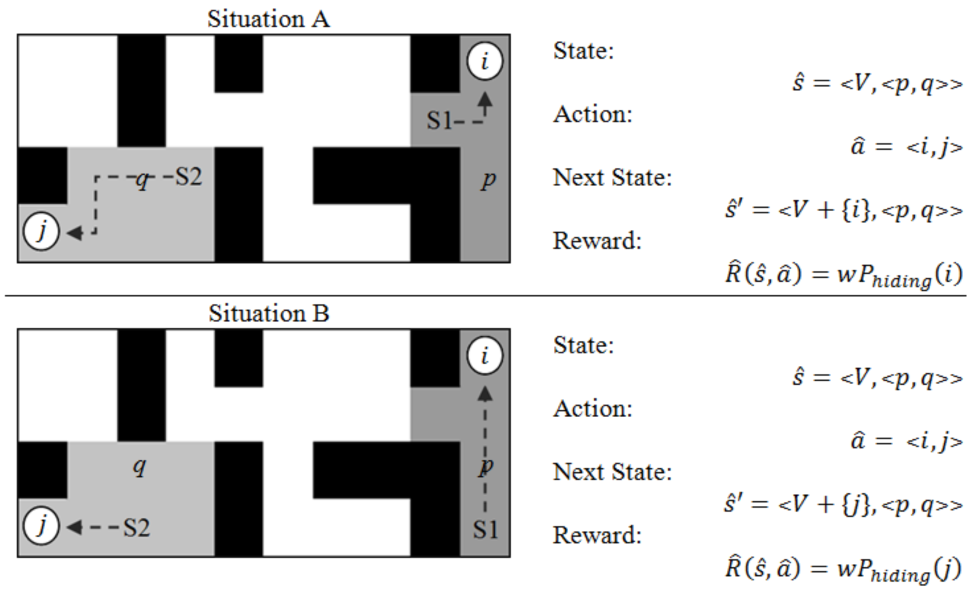


Fig. 5.4. An example of a problematic situation for two seekers S1, S2 when using the first solution. Seekers are in regions with representatives *p*, *q* and taking actions to visit hiding places *i*, *j*, respectively.

The major source of this problem is the unequal time steps required for executing the same joint action for the two seekers. As long as the transition function depends on the time required by the joint actions, the transition state and reward value also depend on the transition time τ of the joint actions. Therefore, we propose another solution that takes the transition time into consideration to avoid the previous drawbacks. The same formulation is used; however, there is a modification in calculating the transition time required by each individual action in the joint action vector. Instead of calculating the transition time required to move from the current location cell to the next hiding destination, the region representative is used as the source instead of the actual location cell of the seeker. With this approach, regardless of the actual location cells of the seekers, the transition time is normalized for all cell cases that reside at the same region. Accordingly, the next transition state and reward is computed based on this calculated transition time. That is why the region representatives play a vital role in the learning model with topological aggregation. In spite of selecting the representatives as the geometric median, there is still an approximation error in the calculated transitions. As mentioned in section 4.2 the transition time τ is included in calculating the estimated Q values via the updating rule (4.4). However the used value of τ is approximated, not the actual transition time. As a result, the estimated Q values will no longer converge to the exact optimal value, but rather converge to slightly higher or lower value depending on the degree of approximation occurred.

5.2.4 Hiding Aggregation

In the hiding aggregation, the objective is to aggregate some mutually close hiding places together into a single target, called hiding spot, and then a seeker deals with the hiding spot as a single destination. For instance, assume a seeker is in location *loc* and there are two possible hiding places that are relatively close to the current location whereas there is another possible hiding place that is relatively far. If the seeker selects one of the close hiding places as his next action, then it would be intuitively reasonable for the other close hiding place to be the consecutive action rather than the far one. In other words, deciding on using one of the near hiding places as the next action implies the decision of the following action to be the other close one. Once the seekers construct the hiding spots, they should exploit this aggregation level to learn over the aggregated hiding spots.

The aggregates or hiding spots are constructed in an earlier stage of learning in which the seekers learn the hiding probability distribution of the hider. After acquiring the sufficient knowledge about the locations of the possible hiding places, the seekers construct hiding spots according to the same two aggregation parameters:

- Spot Size (*maxSize*): The maximum number of hiding places that belong to a single hiding spot. This parameter is important to prevent constructing large hiding spots to preserve reasonable workload balance.
- Spot Diameter (*Dia*): The maximum distance between any two hiding places belonging to the same hiding spot. This parameter reflects the concept of grouping relatively close hiding places.

Fig. 5.5 shows an example of the resulting regions constructed with *maxSize* = 5 and *Dia* = 25. Hiding places with the same mark form a hiding spot. The size of the state space is highly reduced. Without hiding aggregation, the V portion in the state definition takes up to 2^{12} possible values for 12 hiding places. Fortunately with hiding aggregation, it takes up to 2^{8} possible values for 8 possible hiding spots.

The learning model is modified to include the use of hiding aggregation. The impact of the hiding aggregation in the learning process appears in the action definition. Instead of defining the seeker's action as the next unvisited hiding place, it should be the next unvisited hiding spot. After a seeker decides its next action, a hiding spot, it has to visit all the hiding places included in this hiding spot. The internal order of visiting these hiding places (in the same spot) is decided based on the nearest hiding place to its current location. A seeker remains in an intermediate state, i.e. takes ground actions, until all the hiding places in the current hiding spot are visited.

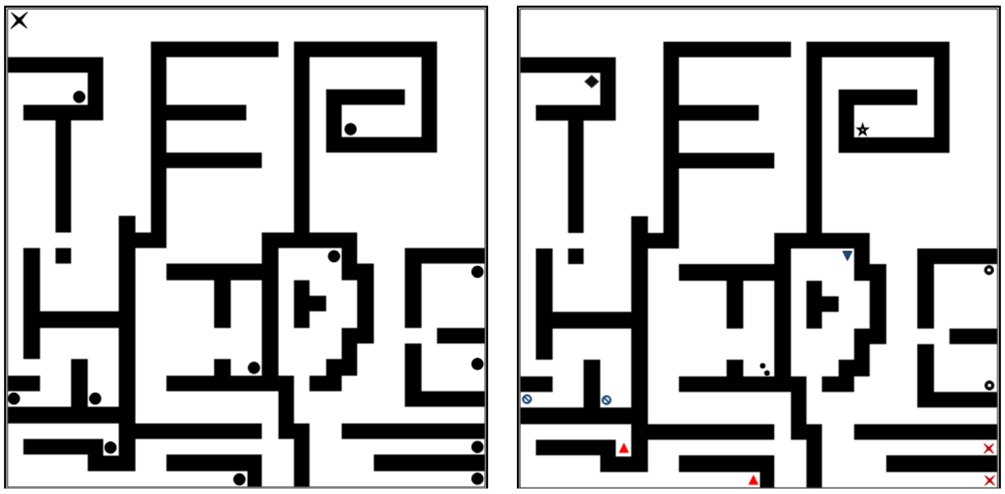


Fig. 5.5. An example of hiding aggregation: *maxSize* = 5, *Dia* = 25. Cells with the same mark forms a hiding spot.

This aggregation approach considers only the distance between the hiding places as an aggregation metric. Consequently, the *Dia* parameter should be carefully selected and tuned for each testing map. Note that the visit sequence of the hiding places aggregated in the same hiding spot is unbreakable, i.e. a seeker will consecutively visit all hiding places of the current hiding spot before visiting another hiding spot. If the *Dia* parameter is not well tuned, the learning process may converge to a poor suboptimal solution. In the scope of this paper, we use manual parameter tuning for each test case. However, some parameter tuning techniques can be adopted. Another source of approximation error is the internal order of visiting the hiding places in each hiding spot. This order considers visiting the nearest hiding places first, however that is not necessarily the optimal strategy. The tuning technique should also consider the difference in the hiding probabilities. We intend to invest these enhancements in future work.

5.3 Experiments

As the problem of space explosion appears with large state and action spaces, we consider testing experiments with large state-action space to exploit the impact of our aggregation technique. Using the same test setting as in Fig. 5.3 and Fig. 5.5, Fig. 5.6 shows the learning progress using the same performance metric *MaxQ* as in section 4.3. It is clear how the learning with aggregation, both topological and hiding aggregation, gives better performance. As the *MaxQ* value that represents the expected total reward value grows more rapidly than the learning without aggregation. That reflects higher convergence rate. Fig. 5.7 shows the resulting seeking trajectories after the same number of learning episodes in both cases. The seeking trajectories in learning with aggregation look more logical than it is in the other case; it has almost no duplicated work, which gives a hint for how good the solution would be. On the other hand, the trajectories without aggregation are much complicated with a lot of duplicated work, e.g. sometimes the same trajectory line is passed more than once.

To get a clear evaluation for the impact of using state aggregation, we need to measure the space reduction achieved by using the proposed aggregation approach. The total number of state-action entries in the *Q*-table reflects the state-action space size; we use the size of the *Q*-table as a metric to study the space reduction accomplished. A set of maps with different characteristics are designed to be used in the experiments as in Fig. 5.8. For example, the *REP* map is divided vertically into three with 12 possible hiding places. The *Spiders* map is diagonally symmetric with 26 hiding places. The last *Rooms* map is more complex with 50% of the map has obstacles arbitrarily distributed. Fig. 5.9 shows the growth in the *Q*-table size over time. Three learning approaches are compared: learning with no aggregation, learning with topological aggregation only, and learning with both topological and hiding aggregation. It is clear how much reduction is achieved in the case of learning with aggregation over the case without aggregation; it is about 10x reduction. As anticipated, the reduction due to the hiding aggregation is much higher than the reduction gained by topological aggregation only, as mentioned in the analysis at section 4.4. These significant results enable hard problems with large state spaces to be solved nearly optimal within a reasonable time and space requirements using the learning approach.

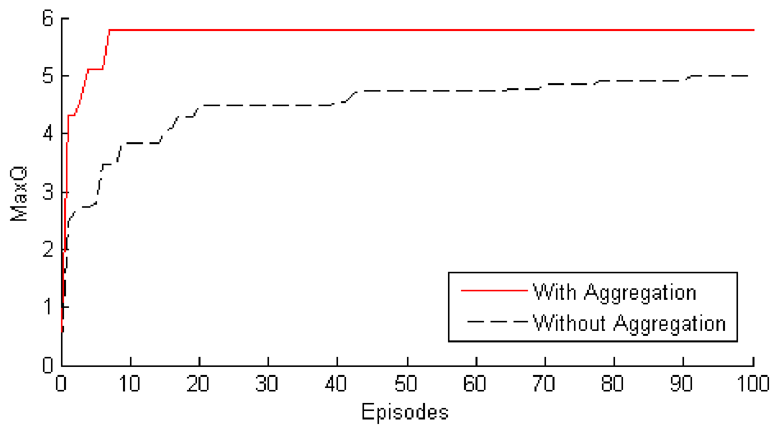


Fig. 5.6. Learning progress with and without aggregation (topological and hiding aggregation).

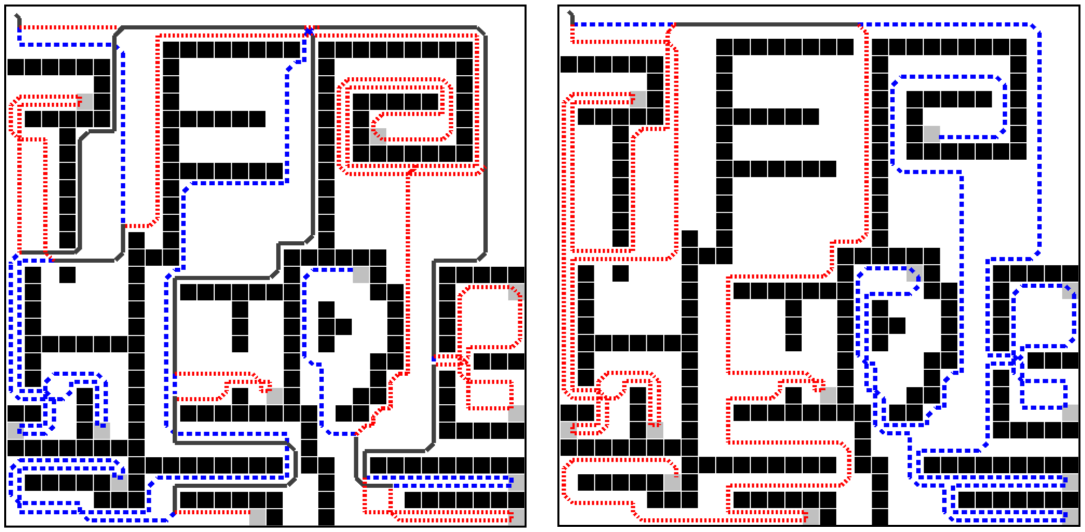
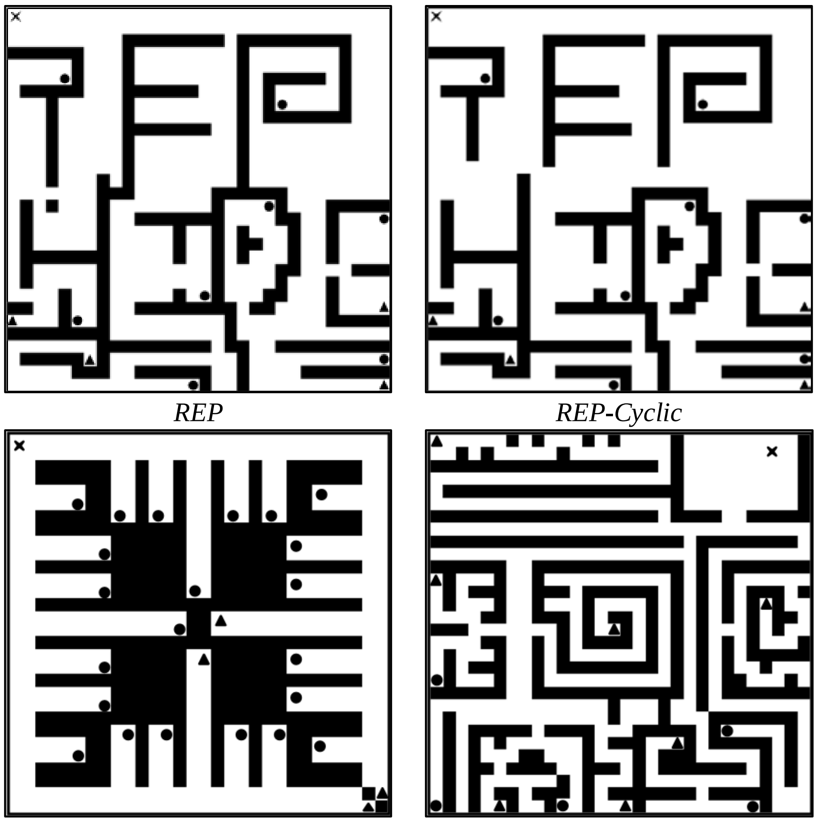

Without aggregation With aggregation

Fig. 5.7. Seeking trajectories with and without aggregation.


REP *REP-Cyclic*

Spiders *Rooms*

Fig. 5.8. Different testing maps.

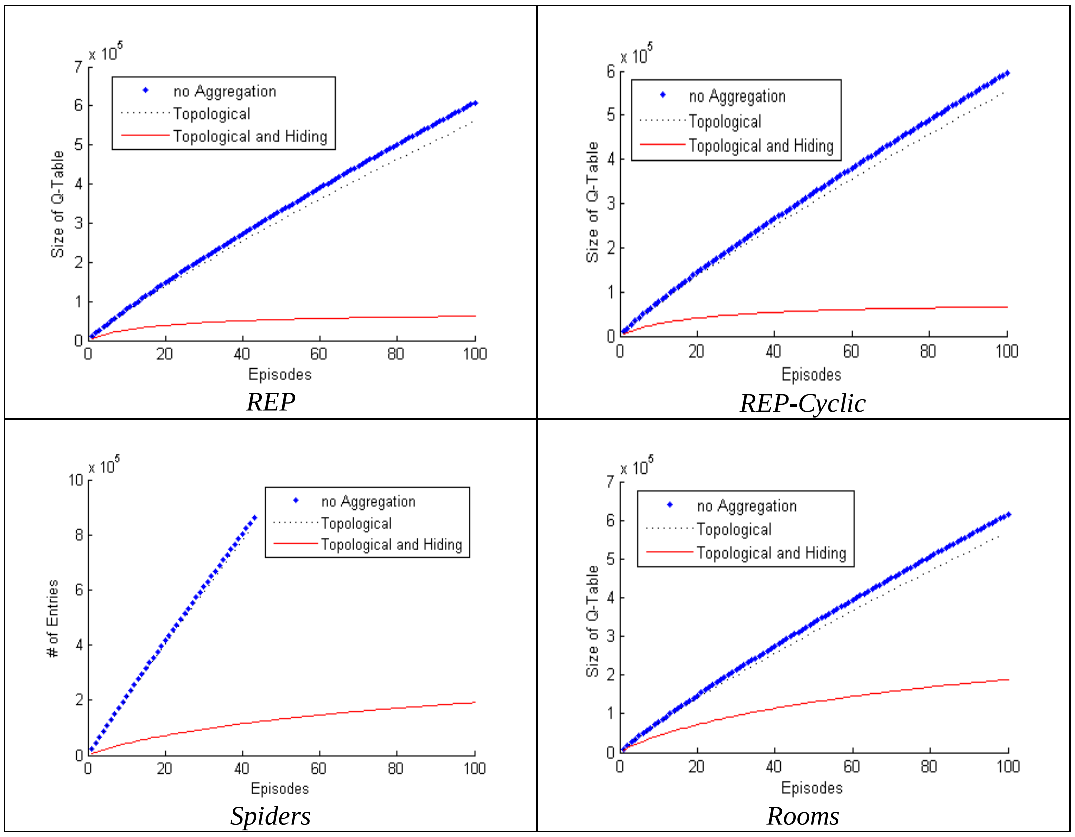


Fig. 5.9. Growth in *Q*-Table size over time for the three learning approaches.

5.4 Section summary

In this section, we tackled the problem of space explosion in our learning model by the state aggregation technique. State aggregation groups similar states that do not have a significant impact on the learning results. They are grouped into one aggregate state that is dealt with as a single state; hence, the state space size is reduced. The idea is used with the territory-division learning model. Two levels of aggregation were proposed: topological aggregation and hiding aggregation. The required modification in the learning model was elaborated accordingly. Then the new learning model was examined on a set of testing maps. The results were compared to the former results without aggregation. The results showed promising performance and high reduction in time and space when using both kinds of aggregation rather than a single kind of aggregation.

6 Conclusion

This paper tackles the problem of multi-seeker's territory division on a Hide-and-Seek game. It discusses a new learning model to solve the problem of territory division that appears in many multi-agent applications such as disaster rescuing, mine detection, robotic surveillance and warehouse systems. Learning is done in a hierarchical scheme using Reinforcement Learning. This hierarchical scheme simplifies the problem by dividing it into several levels. Each level can be solved separately in a bottom-up approach. It is even possible to solve each level with a different technique. In the scope of this paper, we consider only two levels and focus more on the higher level. Although the current learning model assumes that the hiding probability distribution is a priori determined, it would be useful for many applications to consider online estimation. That can be realized maybe by adding a third level of learning at the bottom of the learning hierarchy.

One direct application for our work is the robotic application of mine detection. In which there are multiple robots to scan a given map for landmines. The objective is to divide the scanning effort among the agents to achieve an efficient scanning plan, given some prior uncertain information regarding the distribution of the mines. *Elemental tasks* (tasks in the lower level) correspond to how to reach a candidate mine location from the current location, and these tasks are solved with standard *Q*-learning or any other path planning technique. While composite tasks (tasks in the higher level) correspond to building the scanning trajectories for each robot to achieve an efficient full scan. Our learning model in the higher level can be directly applied using the same definition of state (agents' current locations, and the already detected mine locations), while the action space includes the not scanned locations that might be mine locations. The hiding probability simply reflects the prior knowledge about the distribution of the mines, e.g. if some locations are more likely to contain mines or uniform if no prior knowledge. The reward function can also include information about the terrain, e.g. locations with hard terrain may be given less reward to infer high scanning effort. Another application that can directly use our technique is the robotic vacuum cleaning. This problem fits more our enhanced model with aggregation by considering the rooms as the hiding spots (aggregates). While the hiding probability distribution represents how urgent the room need to be cleaned. And so on, our framework can be applied in various applications, just by defining the two levels of tasks, and the representation of the hiding places and the hiding probability distribution.

As the territory division problem tends to be an *NP-hard* problem, it requires high computational costs. In this paper, we tackle the curse of dimensionality in the learning state space. The learning model is enhanced by the means of *state aggregation* to achieve reduction in time and space requirements. The idea is to group similar states into one aggregate state, and then do learning over the aggregate level, which is much reduced in size. We enhance the learning model using two levels of aggregation, topological aggregation and hiding aggregation. The enhanced model is examined against a set of maps with different characteristics. The results showed an enhanced performance compared to the learning without aggregation. The aggregated model converged faster and achieved up to 10x space reduction. In the scope of the current work, the aggregation parameters are selected by intuitive heuristics and manual trial and error. However, the techniques of parameter tuning should be explored as future work. Another important extension would be to consider the hiding probability distribution while constructing the topological aggregates instead of considering only the topology of the map.

Acknowledgements

This work has been funded by a scholarship grant from Mitsubishi Corporation International Scholarship to E-JUST which is highly acknowledged.

References

Agmon, N., Hazon, N., Kamink, G.A., 2006. Constructing Spanning Trees for Efficient Multi-robot Coverage. In: Proceedings of the 2006 IEEE International Conference on Robotics and Automation, ICRA.

Balas, E., Toth, P., 1985. Branch and bound methods. In: Lawler EL, Lenstra JK, Rinooy Kan AHG & Shmoys DB (eds). The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization, Wiley: Chichester, pp. 361–401.

Bertsekas, Dimitri, P., Castanon, David, A., 1989. Adaptive aggregation methods for infinite horizon dynamic programming. IEEE Transactions on Automatic Control, 34 (6), pp. 589-598.

Bektas, T., 2006. The multiple traveling salesman problem: an overview of formulations and solution procedures. Omega, 34, 209-219.

Choset, H., 2001. Coverage for robotics a survey of recent results. Annals of Mathematics and Artificial Intelligence, 31(1-4), pp. 113–126.

Dijkstra, E.W., 1959. A note on two problems in connexion with graphs. Numerische Mathematik 1, pp. 269–271.

Galceran, E., Carreras, M., 2013. A survey on coverage path planning for robotics. Robotics and Autonomous Systems, Vol. 61, Issue 12, pp. 1258–1276

Gavish, B., Srikanth, K., 1986. An optimal solution method for large-scale multiple traveling salesman problems. Operations Research, Vol. 34, No. 5, pp. 698–717.

Gilbert, K.C., Hofstra, R.B., 1992. A new multiperiod multiple traveling salesman problem with heuristic and application to a scheduling problem. Decision Sciences, Vol. 23, pp. 250–9.

Gunady, M.K., Gomaa, W., 2012. Reinforcement learning generalization using state aggregation with a maze-solving problem. Proceedings of IEEE Japan-Egypt Conference on Electronics, Communications and Computers (JEC-ECC), pp. 157–162.

Gunady, M.K., Gomaa, W., Takeuchi, I., 2012. Multi-agent Task Division Learning in Hide-and-Seek Games. Artificial Intelligence: Methodology, Systems, and Applications (AIMSA), LNCS, Vol. 7557, pp. 256–265.

Held, M., Karp, R.M., 1962. A Dynamic Programming Approach to Sequencing Problems, Journal of the Society for Industrial and Applied Mathematics 10 (1): 196–210, doi: 10.1137/0110015.

Johnson, D.S., McGeoch, L.A., 1995. The Traveling Salesman Problem: A Case Study in Local Optimization, November 20, 1995.

Kaelbling, L.P., Littman, M.L., Moore, A.W., 1996. Reinforcement learning: A survey. Arxiv preprint cs/9605103, pp. 237-285.

Lenstra, J.K., Rinnooy Kan, A.H.G., 1974. Some Simple Applications of the Travelling Salesman Problem. BW 38/74, Stichting Mathematisch Centrum, Amsterdam.

Leng, J., Jain, L., Fyfe, C., 2007. Convergence Analysis on Approximate Reinforcement Learning. KSEM 2007, Z. Zhang and J. Siekmann (Eds.), LNAI 4798, pp. 85–91.

Modares, A., Somhom, S., Enkawa, T., 1999. A self-organizing neural network approach for multiple traveling salesman and vehicle routing problems. International Transactions in Operational Research, Vol. 6, pp. 591–606.

Marthi, B., 2007. Automatic shaping and decomposition of reward functions. Proceedings of the 24th international conference on machine learning, pp. 1-15.

Ralphs, T.K., 2003. Parallel branch and cut for capacitated vehicle routing. Parallel Computing, Vol. 29, pp. 607–29.

Rekleitis, I., New, A., Rankin, E., Choset, H., 2009. Efficient boustrophedon multirobot coverage: an algorithmic approach, Annals of Mathematics andArtificial Intelligence 52, pp. 109–142.

Schrijver, A., 1960. On the history of combinatorial optimization.

Sutton, R.S., 1988. Learning to predict by the methods of temporal differences. Machine Learning, Vol. 3, No. 1, pp. 9-44.

Singh, S.P., 1992a. The Efficient Learning of Multiple Task Sequences. Advances in Neural In-formation Processing Systems 4, pp. 251–258.

Singh, S.P., 1992b. Transfer of learning by composing solutions for elemental sequential tasks. Machine Learning.

Singh, S., Jaakkola, T., 1995. Reinforcement learning with soft state aggregation. Advances in neural information, 1995.

Sutton, R.S., Barto, A.G., 1998. Reinforcement Learning: An Introduction. MIT Press, Cam-bridge, MA.

Singh, S., Jaakkola, T., Littman, M., 2000. Convergence results for single-step on-policy reinforcement-learning algorithms. Machine Learning, No. 1998, pp. 287-308.

Sofge, D., Schultz, A., De Jong, K., 2002. Evolutionary computational approaches to solving the multiple traveling salesman problem using a neighborhood attractor schema. Lecture notes in computer science, Vol. 2279, pp. 51–60.

Tang, L., Liu, J., Rong, A., Yang, Z., 2000. A multiple traveling salesman problem model for hot rolling scheduling in Shangai Baoshan Iron & Steel Complex. European Journal of Operational Research, Vol. 124, pp. 267–82.

Tang, L., An, B., Cheng, D., 2007. An agent reinforcement learning model based on neural networks. Bio-Inspired Computational Intelligence and Applications, pp. 117–127.

Vakhutinsky, I.A., Golden, L.B., 1994. Solving vehicle routing problems using elastic net. Proceedings of the IEEE international conference on neural network, pp. 4535–40.

Wacholder, E., Han, J., Mann, R.C., 1989. A neural network algorithm for the multiple traveling salesmen problem. Biology in Cybernetics, Vol. 61, pp. 11–9.

Watkins, C.J.C.H., Dayan, P., 1992. Q-learning. Machine learning, Vol. 8, No. 3, pp. 279–292.

Yu, Z., Jinhai, L., Guochang, G., Rubo, Z., Haiyan, Y., 2002. An implementation of evolutionary computation for path planning of cooperative mobile robots. Proceedings of the fourth world congress on intelligent control and automation, Vol. 3, pp. 1798–802.