An Efficient Approach to Model-Based Hierarchical Reinforcement Learning

- **Yunqiu Xu**

---

1. Introduction

- Current HRL can not handle real world problems:
  - Multiple tasks
  - Changing subgoals
  - Uncertain subtask specifications
- Two limitations of MAXQ-based (**pre-defined task hierarchy**) methods
  - All of the tasks / subtasks need to be clearly specified
  - Even similar subtasks have to be learned separately
- Our work: context-sensitive reinforcement learning
  - Exploit common knowledge in subtasks $\rightarrow$ learn transition dynamics effiently
  - Actively evaluate different subtasks as execution choices
  - Based on simulation

2. Problem Setup

- Task $T_i = \{I_i, G_i, F_i, A_i, T_i, R_i\}$
  - I_i : input set
  - G_i : goal, terminating states
  - F_i : relevent features
  - A_i : actions
  - T_i : transition functions
  - R_i : reward functions
  - 这个比我之前看得 Task 定义要复杂不少
  - Well-defined MDP : $\{F_i, A_i, T_i, R_i\}$

- Fragment $\{F_j, A_j, T_j\}$
    - No goal states or local reward functions
    - Can describe **similar multiple tasks** $\rightarrow$ share same transition function, but differ in goals
    - How to generate: combine tasks with same F_i and A_i
    - How it be used: facilitate efficient learning of task transition functions
    - 例如 Fig 4 中 Get 和 Put 两个任务都可以归于一个 Fragement

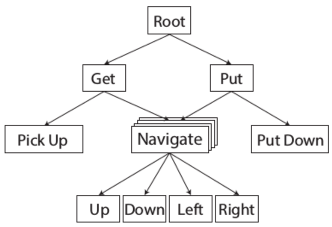

Figure 3: MAXQ hierarchy

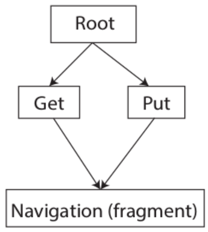

Figure 4: CSRL hierarchy

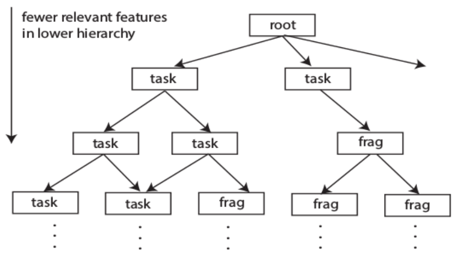

Figure 5: A general CSRL hierarchy

- General CSRL:
    - Node : task or fragment
    - Different from MAXQ:
        - A task can be decomposed as fragments / smaller tasks
        - Actions are defined inside each node $\rightarrow$ primitive actions won't appear as the leaf nodes
        - F of a node is also the subset of F of its parent
    - **Fragments allow us specify tasks without goal states**
        - 这里按照我的理解就是把两个动作/特征相似的任务归为一类
        - 例如抓取和放置都可以归类为定位
        - 因此学会抓取之后很快就能学会放置

3. Algorithm

3.1 Overview of CSRL

Algorithm 1 CSRL Algorithm

1: **Input:** m, F_i for all tasks i
2: $s \leftarrow s_0, X_a \leftarrow \{i | a \in A_i\}$
3: **for all** components k **do**
4:     $X_{C_k} \leftarrow \{i | C_k \subseteq F_i\}$ // tasks using C_k
5:     **for all** actions a **do**
6:         $\mathcal{P}_{k,a} \leftarrow \bigcap_{i \in X_{C_k} \cap X_a} F_i$
7:         $n(\mathcal{P}_{k,a}, a) \leftarrow 0; P(\cdot | \mathcal{P}_{k,a}, a) \leftarrow \varnothing$
8:     **end for**
9: **end for**
10: **while** $s \notin$ terminal state **do**
11:     **for all** components k, action a **do**
12:         **if** $n(\mathcal{P}_{k,a}, a) < m$ **then**
13:             $\hat{P}(C_k^f | \mathcal{P}_{k,a}, a) \leftarrow 1$
14:         **else**
15:             $\hat{P}(\cdot | \mathcal{P}_{k,a}, a) \leftarrow P(\cdot | \mathcal{P}_{k,a}, a)$
16:         **end if**
17:     **end for**
18:     **if** current task is completed or no task selected **then**
19:         ConstructSMDP(s)
20:         Solve previous SMDP to get next task i.
21:     **end if**
22:     Construct model for task i using Alg 1.
23:     Solve model for task i to get a task policy π_i.
24:     Execute $(s, \pi_i(s)) \rightarrow s'$
25:     **for all** components k **do**
26:         $n(\mathcal{P}_{k,a}, a) \leftarrow n(\mathcal{P}_{k,a}, a) + 1$
27:         Update $P(\cdot | \mathcal{P}_{k,a}, a)$ with s, s'
28:     **end for**
29:     $s \leftarrow s'$
30: **end while**

- m : exploration threshold
- $P_{k,a} = Parent(C_k, a)$: the parent of C_k with action a
- Line 6 : init $P_{k,a}$
- $n(P_{k,a}, a)$: exploration count
    - If smaller than m $\rightarrow$ transits to a fictitious component C_k^f
    - Else $\rightarrow$ update probability values

- Line 18-21 : select a new task when a task is finished

3.2 Select a new task

Algorithm 2 ConstructSMDP(current_state)

```
state_queue.enqueue(current_state)
while state_queue not empty do
    s = state_queue.dequeue
    if s is new then
        for all tasks i do
            [succ, reward, dist] = SimulateTask(s, i)
            if ¬succ then
                R(s, i) ← R_max
                P(s|s, i) ← 1
            else
                R(s, i) ← reward
                P(·|s, i) ← dist
            end if
            for all states s′ in dist do
                state_queue.enqueue(s′)
            end for
        end for
    end if
end while
```

- Model task selection as an S(semi)MDP
- Recursively : given a task policy for any node, we can construct the task selection SMDP at the parent node

3.3 Task Simulation

- Why simulate the task?
  - As the transition function has been computed for each node in the hierarchy
  - The parameters can be more efficiently estimated by simulating

- What do we simulate?
  - The effect of executing the task policy on the task' s parent node' s transition function
- How to simulate?

Algorithm 3 SimulateTask(s, i)

$distribution \leftarrow \varnothing$,$average\_reward \leftarrow 0$
for $simulation\_num$: 1 to NUM_SIM **do**
    $reward \leftarrow 0$; $steps \leftarrow 0$
    while True **do**
        $a \leftarrow \pi_i(s)$; $reward \leftarrow reward + R_i(s, a)$
        $s \leftarrow$ SampleRootTransition (s, a)
        $R(s, i) \leftarrow R_{\max}$
        if $steps >$ NUM_STEPS or s is fictitious **then**
            return [False, Null, $\varnothing$]
        end if
        if s is goal state **then**
            Update $average\_reward$ with $reward$
            Update $distribution$ with s as terminal state
            Update $distribution$ with duration to complete
            break
        end if
    end while
end for
return [True, $average\_reward$, $distribution$]

- Given the policies of the child tasks, CSRL can simulate the results of executing the task on the root node

4. Experiments

- Robot pickup and place : does not require task selection
- Pickup and place with two objects
- Household robot experiment

- Requires multiple levels of reasoning
- Cannot be solved using existing methods due to incomplete problem specification at the lower level

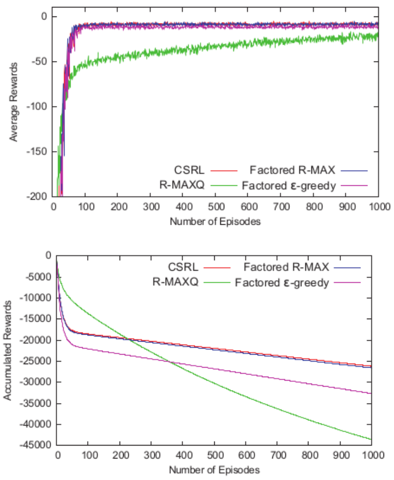

Figure 6: Robot pickup and place

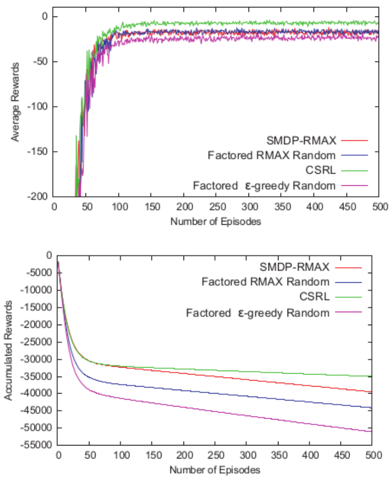

Figure 7: Pickup and place: 2 objects

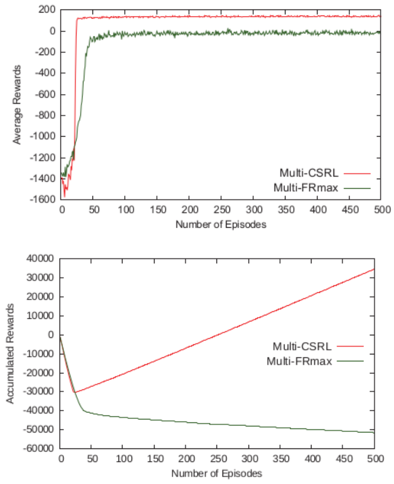

Figure 8: Household assistive robot

5. Conclusion

- Task learning mechanism: learn both task and global transition dynamics
- Hierarchical execution mechanism: handle task selection by formulating and solving the underlying SMDP
- Limitation:
    - Specify relevant features manually
    - Can not build sub-tasks automatically
    - No transfer learning : in the future we can try to perform transfer learning for those with similar hierarchy