Less Is More: Refining Datasets for Offline Reinforcement Learning with Reward Machines

Haoyuan Sun
School of Computer Science and Technology,
University of Science and Technology of China,
Hefei, Anhui, China
sunhaoyuan@mail.ustc.edu.cn

Feng Wu
School of Computer Science and Technology,
University of Science and Technology of China,
Hefei, Anhui, China
wufeng02@ustc.edu.cn

ABSTRACT

Offline reinforcement learning (RL) aims to learn a policy from a fixed dataset, without further interactions with the environment. However, offline datasets are often very noisy, which consist of large quantities of sub-optimal or task-agnostic trajectories. Therefore, it is very challenging for offline RL to learn an optimal policy from such datasets. To address this, we use reward machines (RM) to encode human knowledge about the task and refine datasets for offline RL. Specifically, we define the event-ordered RM to label offline datasets with RM states. Then, we further use the labeled datasets to generate refined datasets, which is smaller but better for offline RL. By using the RM, we can decompose a long-horizon task into easier sub-tasks, inform the agent about their current stage along task completion, and guide the offline learning process. In addition, we generate counterfactual experiences by RM to guide agent to complete each sub-task. Experimental results in the D4RL benchmark confirm that our method achieves better performance in long-horizon manipulation tasks with sub-optimal datasets.

KEYWORDS

Offline Reinforcement Learning; Reward Machines; Dataset Refinement; Counterfactual Reasoning

ACM Reference Format:
Haoyuan Sun and Feng Wu. 2023. Less Is More: Refining Datasets for Offline Reinforcement Learning with Reward Machines. In *Proc. of the 22nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2023), London, United Kingdom, May 29 – June 2, 2023*, IFAAMAS, 9 pages.

1 INTRODUCTION

Reinforcement learning (RL), especially when combines with deep neural networks, has achieved great success in various applications, ranging from games to robotics. However, applying RL to real-world scenarios is still challenging because exploration and interaction with real-world environment is often costly or risky. For example, a self-driving car learning in real world requires human supervision and safety checks, which may be dangerous or time-consuming. In such settings, it is desirable to learn from previously collected data. Offline RL aims to address the problem of learning effective policies entirely from offline datasets, without online interaction. To date, several offline RL methods [6, 10–12, 14, 27, 29] have been proposed and successfully tested in many domains.

Feng Wu is the corresponding author.

Proc. of the 22nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2023), A. Ricci, W. Yeoh, N. Agmon, B. An (eds.), May 29 – June 2, 2023, London, United Kingdom. © 2023 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). All rights reserved.

Unfortunately, offline RL is generally more challenging than its online counterpart and its performance heavily depends on the quality of datasets. Firstly, the long-horizon task, which is a common challenge in RL, becomes even more difficult in offline settings. This is because the agent is not allowed to explore further. Secondly, there are often many sub-optimal or task-agnostic trajectories in offline datasets. This is because the demonstrations from human experts are very costly and offline datasets are often generated by some poor behavior policies. In offline RL, the quality of the datasets is critical for learning. To date, most of the offline RL methods [10, 11] still fail to learn a good policy from low-quality datasets. It is worth noting that the datasets can be very large but not necessarily high-quality in many real-world applications.

Motivated by mining and metal refinement, the datasets should be able to be refined with some carefully designed processes using human knowledge. Intuitively, this will make the datasets smaller and more suitable for offline RL. Indeed, decomposing a long-horizon task into a series of easier sub-tasks is a key capability of our human beings. Moreover, people also tend to focus on experiences that are useful for the specific task. For example, we perform the task of opening a door in two stages, i.e., 1) reaching to the door handle first and 2) then pulling the door. We are also very sensitive to tell which experiences are relevant to door opening. Now, the key question is how such human knowledge can be effectively used to refine the datasets of offline RL.

In this paper, we use *reward machine* (RM) to encode task-specific human knowledge. Specifically, RM is a finite-state machine used to define task dependent on high-level events [8]. In more details, it encodes high-level knowledge into its RM states to expose the structure of the reward function. Furthermore, RM allows the agent to decompose a long-horizon task into several stages and learn sets of policies for each stage of the whole task. With high-level knowledge encoded in RM, we can distinguish experiences that are useful for the task from low-quality offline datasets. Besides, RM is very intuitive for us to build as long as we can define the high-level events of a given task.

With the advantage of RM, we propose a novel approach to refine the original datasets in order to generate smaller but better datasets for offline RL. Specifically, we first introduce a special type of RM that can encode the history of high-level events in a trajectory. Then, we build this RM with human knowledge about the task and use it to label trajectories in the datasets. Giving this, we split them into sub-trajectories according to the labeled RM states. With the sub-trajectories, we compute the best order of high-level events of the task and select sub-trajectories aligned with the order. Furthermore, we relabel datasets with counterfactual experiences by RM to make better use of the selected trajectories. Intuitively, this is very similar to the mineral processing, where the crude ores are broken into small pieces and then separate the valuable minerals from the waste rock. Using the refined dataset, we train a policy with state-of-the-art offline RL methods with RM states to complete the whole task. Finally, we conduct our experiments in the common offline RL benchmark — D4RL[5]. Experimental results show our method achieves better performance than baselines on all the tasks. Additionally, we illustrate that our method can generate smaller but better datasets for offline RL.

Our main contributions are summarized as follows: 1) we use RM to encode human knowledge and propose a method to refine the datasets for offline RL and 2) we show the effectiveness of our method on the challenging tasks in the D4RL benchmark. To the best of our knowledge, we are the first to use RM to refine the datasets and demonstrate that "less is more" for offline RL.

2 BACKGROUND

2.1 Offline Reinforcement Learning

The *reinforcement learning* (RL) problem is formulated in the context of a *Markov decision process* (MDP), defined as a tuple $\mathcal{M} = \langle S, A, r, p, \gamma \rangle$, where S is a state space, A is an action space, $r : S \times A \times S \to \mathbb{R}$ is a reward function, $p(s'|s,a)$ is the transition probability distribution, and $\gamma \in (0,1]$ is a discount factor. At every time step t, the agent observes the current state $s_t \in S$ and executes an action $a_t \in A$ following a policy $\pi(a_t|s_t)$, then transitions to a new state $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$ and receives a reward $r(s_t, a_t, s_{t+1})$. The goal of RL is to learn an optimal policy π^* maximizing the expected sum of discounted future rewards by trial-and-error [24].

In contrast to online RL, offline RL trains policy on previously collected data without any additional interaction with the environment. Given a fixed dataset $D = \{(s,a,r,s')\}$, offline RL minimizes the Bellman error derived from the action-value Bellman equation:

$$L(\theta) = \mathbb{E}_{(s,a,r,s') \sim D}[(r + \gamma \max_{a'} Q_{\hat{\theta}}(s', a') - Q_\theta(s,a))^2] \quad (1)$$

where Q_θ is a parameterized Q-function, $Q_{\hat{\theta}}$ is a target network, and the policy is defined as $\pi(s) = \arg\max_a Q_\theta(s,a)$. However, since some state-action pairs (s', a') are often not in D, *out-of-distribution* (OOD) actions a' can produce erroneous values for $Q_{\hat{\theta}}(s', a')$ in the above objective, leading to overestimation as the policy is defined to maximize the (estimated) Q-value.

There are many methods proposed to avoid OOD actions. Among them, *implicit Q-learning* (IQL) [10], which is the leading method for offline RL, solves this issue by estimating the maximum Q-value over actions that are in the support of the data distribution. Formally, they use a value function V_ψ as the target, such that:

$$L(\theta) = \mathbb{E}_{(s,a,r,s') \sim D}[(r + \gamma V_\psi(s') - Q_\theta(s,a))^2] \quad (2)$$

and V_ψ approximates an expectile only with respect to the action distribution, leading to the following loss:

$$L(\psi) = \mathbb{E}_{(s,a) \sim D}[L_2^\tau(Q_{\hat{\theta}}(s,a) - V_\psi(s))] \quad (3)$$

where $L_2^\tau(u) = |\tau - \mathbb{1}(u < 0)|u^2$ and τ is a hyper-parameter. Then, it extracts the policy π_ϕ using advantage weighted regression [16]:

$$L(\phi) = \mathbb{E}_{(s,a) \sim D}[\exp(\beta(Q_{\hat{\theta}}(s,a) - V_\psi(s))) \log \pi_\phi(a|s)] \quad (4)$$

where $\beta \geq 0$ is an inverse temperature.

2.2 Reward Machines

A *reward machine* (RM) is a tuple $\mathcal{R} = \langle U, u_I, F, \Sigma, \delta_u, \delta_r \rangle$, where U is a finite set of states, $u_I \in U$ is an initial state, F is a finite set of terminal states (where $F \subseteq U$), Σ is a finite set of environment events, $\delta_u : U \times \Sigma \to U$ is the state-transition function, and $\delta_r : U \times U \to \mathbb{R}$ is the reward transition function. RM is initially introduced by [8]. Here, we adapt the RM's definition slightly to better suit the offline RL setting, which is similar to [15].

When paired with a MDP, the RM uses a labeling function $L : S \times A \times S \to 2^\Sigma$ to map from a MDP transition (s,a,s') to a set of environment events. If the agent executes action a_t to transition from state s_t to s_{t+1} in the MDP, then the labeling function L outputs a set of environments events that happens in MDP transition (s_t, a_t, s_{t+1}), allowing it to capture scenarios in which multiple events occur concurrently. In such a scenario, the events are passed as a sequence to the RM in no particular order [15].

3 MAIN METHOD

In this section, we propose our method, the framework of which is shown in Figure 1. The basic idea is to decompose the long-horizon task into a sequence of easier events by reward machines. To this end, we first introduce a new type of RM that can capture the order of event history. Then, we use this RM to label the trajectories in the offline dataset with RM states. After that, we split the trajectories into sub-trajectories according to the transition of RM state and properly group them. According to the result of grouping, the best order of the events is generated to complete the task. Given this, sub-trajectories consistent with the order are selected to form the new dataset. Additionally, the counterfactual experiences are generated to relabel the new dataset. Finally, the newly generated dataset is used to learn a policy using offline method (i.e., IQL).

Here, we assume that a set of high-level events that describe the long-horizon task is defined as prior knowledge. Specifically, for a given task E and a properly defined environments events $\Sigma = \{e_i\}_{i=1}^n$, the task E is completed if and only if all of the events in Σ occur. Such events can be naturally obtained for many tasks. For example, in the Franka Kitchen domain [7], the whole task may be to: 1) open the microwave, 2) place a kettle on the burner, 3) turn the overhead light on, and 4) open the sliding cabinet door. In this case, Σ can be defined as $\{e_i\}_{i=1}^4$, where e_1 means "microwave is opened", e_2 means "kettle is on the burner", e_3 means "light is turned on" and e_4 means "sliding cabinet door is opened". The whole task E is completed in a trajectory if and only if all of events $\Sigma = \{e_i\}_{i=1}^4$ occur, which means the four sub-tasks are all completed.

In practice, a feature detector $\Phi : S \to 2^\Sigma$ that maps the state s into a set of events is commonly used to detect the events in Σ [3]. For instance, in the Franka Kitchen domain, $\Phi(s) = \{e_1, e_3\}$ means that in state s, e_1 and e_3 have occurred. Note that sensors must be deployed to detect whether microwave is opened (e_1) or light is turned on (e_3). Hence, the implementation of feature detectors depends on the specific domains.

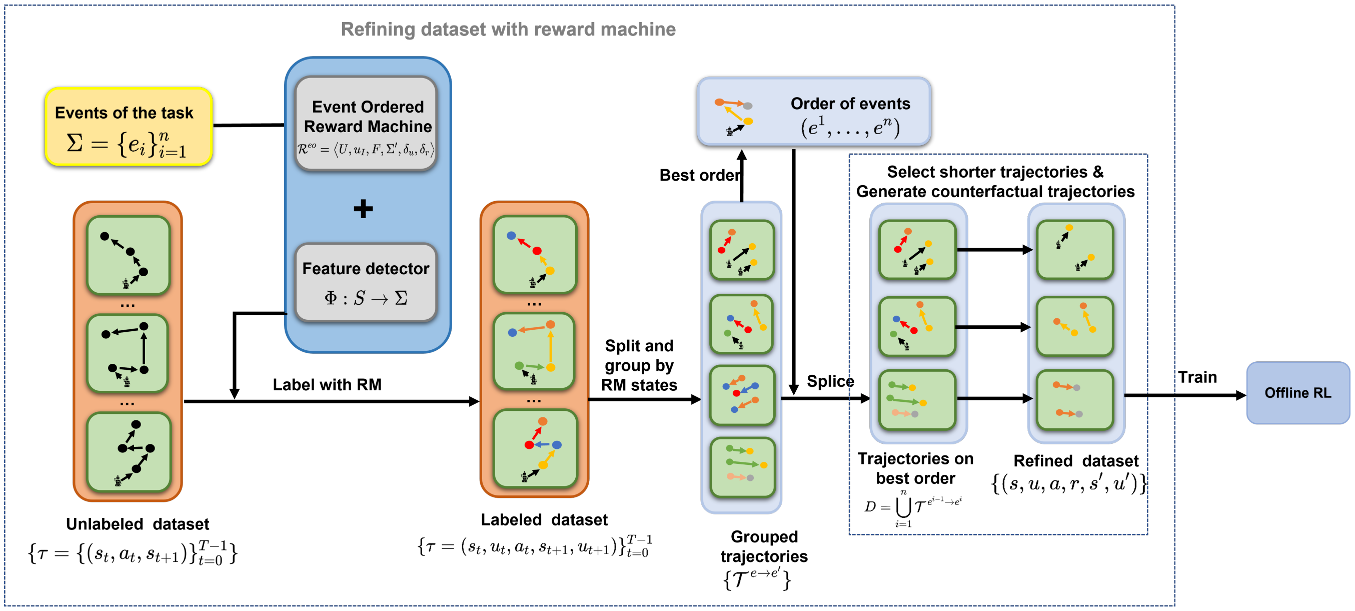

Refining dataset with reward machine
Events of the task
$\Sigma = \{e_i\}_{i=1}^n$
Event Ordered Reward Machine
$\mathcal{R}^{eo} = \langle U, u_I, F, \Sigma', \delta_u, \delta_r \rangle$
Feature detector
$\Phi : S \to \Sigma$
Label with RM
Unlabeled dataset
$\{\tau = \{(s_t, a_t, s_{t+1})\}_{t=0}^{T-1}\}$
Labeled dataset
$\{\tau = (s_t, u_t, a_t, s_{t+1}, u_{t+1})\}_{t=0}^{T-1}$
Split and group by RM states
Grouped trajectories
$\{\mathcal{T}^{e \to e'}\}$
Best order
Order of events
$(e^1, \dots, e^n)$
Splice
Select shorter trajectories & Generate counterfactual trajectories
Trajectories on best order
$D = \bigcup_{i=1}^n \mathcal{T}^{e^{i-1} \to e^i}$
Refined dataset
$\{(s, u, a, r, s', u')\}$
Train
Offline RL

Figure 1: The basic framework of our method.

3.1 Event-Ordered Reward Machines

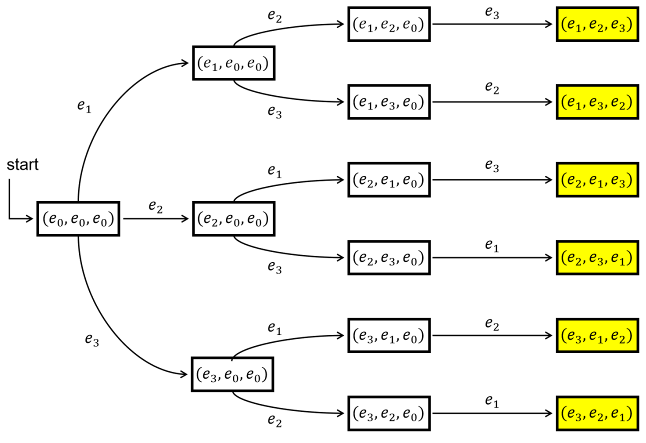
In many tasks, the order of events is critical for learning a policy. In the original RM, the labeling function only maps a transition to a set of events, without considering their occurrence order. To capture the order of event history in a trajectory, we extend the original RM to event-ordered RM defined as follows.

Definition 3.1. Given a set of events $\Sigma = \{e_i\}_{i=1}^n$, event-ordered RM is defined as a tuple $\mathcal{R}^{eo} = \langle U, u_I, F, \Sigma', \delta_u, \delta_r \rangle$, where:

- U is a finite set of states and $u \in U$ is a n-dimension vector $u = (u^1, u^2, \dots, u^n)$, where $u^t \in \Sigma', 1 \leq t \leq n$ is an occurred event. For $\forall 1 \leq t \leq n$, if $u^t = e_0$, then all of its successors $u^{t'} = e_0, t \leq t' \leq n$; if $u^t \neq e_0$, then all of its predecessors $u^{t'} \neq e_0, 1 \leq t' \leq t$, and any two predecessors $u^{t_1} \neq u^{t_2}, 1 \leq t1 \neq t2 \leq t$ are different.
- $u_I = (u^1, u^2, \dots, u^n)$ is the initial state, where none of the events occur, i.e., $u^t = e_0, 1 \leq t \leq n$.
- F is the set of terminal states. A state $u = (u^1, u^2, \dots, u^n) \in F$ if and only if all events occurred, i.e., $u^t \neq e_0, 1 \leq t \leq n$.
- $\Sigma' = \Sigma \cup \{e_0\}$ is the set of events, where e_0 represents none of events in Σ occur.
- δ_u is the transition function and δ_r is the reward function.

The transition function $\delta_u : U \times \Sigma' \to U$ is defined as follows. Given a state $u = (u^1, u^2, \dots, u^n) \in U$ and a event $e \in \Sigma$, we have:

(1) if no event occurs, i.e., $e = e_0$, then the next state u' transits to its current state u, i.e., $u' = \delta_u(u, e) = u$.
(2) if some event occurs, i.e., $e = e_i, 1 \leq i \leq n$, then:
  (a) if this event occurred before, i.e., $\exists 1 \leq t \leq n, u^t = e_i$, then the next state remains the same, i.e., $u' = \delta_u(u, e) = u$.
  (b) if this event is a new event, i.e., $\forall 1 \leq t \leq n, u^t \neq e_i$, then it transits to a new state u', which is a copy of u except that the k-th element of u' is set to e. Here, $k = \min_{1 \leq k \leq n}\{u^k = e_0\}$ is the first element of u that equals to e_0.



$$\begin{tikzcd}
 & & (e_1,e_2,e_0) \arrow[r, "e_3"] & (e_1,e_2,e_3) \\
 & (e_1,e_0,e_0) \arrow[ru, "e_2"] \arrow[r, "e_3"'] & (e_1,e_3,e_0) \arrow[r, "e_2"] & (e_1,e_3,e_2) \\
 & & (e_2,e_1,e_0) \arrow[r, "e_3"] & (e_2,e_1,e_3) \\
\text{start} \arrow[r] & (e_0,e_0,e_0) \arrow[uu, bend left, "e_1"] \arrow[r, "e_2"] \arrow[dd, bend right, "e_3"'] & (e_2,e_0,e_0) \arrow[u, "e_1"] \arrow[d, "e_3"'] & \\
 & & (e_2,e_3,e_0) \arrow[r, "e_1"] & (e_2,e_3,e_1) \\
 & & (e_3,e_1,e_0) \arrow[r, "e_2"] & (e_3,e_1,e_2) \\
 & (e_3,e_0,e_0) \arrow[ru, "e_1"] \arrow[r, "e_2"'] & (e_3,e_2,e_0) \arrow[r, "e_1"] & (e_3,e_2,e_1)
\end{tikzcd}$$

Figure 2: Example of RM with $n = 3$ and $\Sigma = \{e_1, e_2, e_3\}$.

The reward function $\delta_r : U \times U \to \mathbb{R}$ is defined as:

$$\delta_r(u, u') = \begin{cases} 0, & u \neq u' \\ -1, & \text{others} \end{cases}$$

Note that the states of RM model the completion of the events in Σ and their occurred order. Figure 2 shows a graphical representation of event-ordered RM where $\Sigma = \{e_i\}_{i=1}^n, n = 3$. As shown, every node in the graph is a state of the machine. Among them, the node labeled with "start" (e_0, e_0, e_0) is the initial state and the yellow nodes are the terminal states. The initial state $u_I = (e_0, e_0, e_0)$ means that none of events in Σ occurs. The state $u = (e_2, e_1, e_0)$ means that events e_2, e_1 have occurred in that order and e_3 has

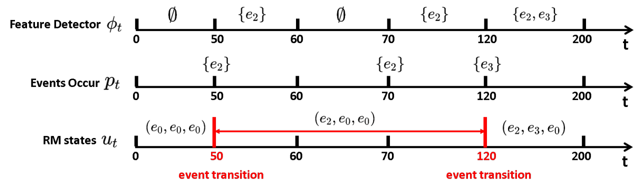


Figure 3: Trajectory labeling in Example 3.1

not occurred yet. Notice that (e_1, e_2, e_0) is different from (e_2, e_1, e_0). Though they both complete the $\{e_1, e_2\}$, they represent different orders of occurrence of $\{e_1, e_2\}$. The terminal states mean that all of the events $e \in \Sigma$ have occurred. The different states represent different orders of occurrence of Σ.

Here, edges labeled by $e \in \Sigma' = \Sigma \cup \{e_0\}$ between u, u' represent the transition $\delta_u(u, e) = u'$. For easy visualization, self-loop transitions are not shown in the graph. Take state $u = (e_1, e_0, e_0)$ for example. If $e = e_0$, which means nothing occurs, we have $u' = \delta_u(u, e_0) = u$. If $e = e_1$, which means event e_1 occur, we also have $u' = \delta_u(u, e_1) = u$. This is because the event e_1 has occurred before and the state does not transit for e_1 occurring in the second time. If $e = e_2$ or $e = e_3$, the RM state transits to (e_1, e_2, e_0) or (e_1, e_3, e_0) respectively, which means a new event occurs.

With this RM, we can have access to history information of a state, which is important for understanding and dealing with a trajectory in the offline dataset. Later we will show how to use this RM to label and refine offline datasets.

3.2 Labeling Offline Datasets with RM States

With event-ordered RM introduced above, we can label the offline dataset by the RM states. Specifically, given a set of events Σ and a feature detector $\Phi : s \longrightarrow 2^{\Sigma}$, our goal is to label a trajectory $\{(s_t, a_t, s_{t+1})\}_{t=0}^{T-1}$ to the one with RM states $\{(s_t, u_t, a_t, s_{t+1}, u_{t+1})\}_{t=0}^{T-1}$, where T is the length of the trajectory. In the original RM [8], a labeling function $L : S \times A \times S \longrightarrow 2^{\Sigma}$ is defined to indicate what occurred in a transition (s, a, s'). However, it is difficult to directly define such a labeling function in complex tasks. For example, in the Franka Kitchen domain, the robot arm has 9-DoF, and it is hard to know what the robot is doing given only a transition. As aforementioned, we address this problem by a feature detector [3].

Algorithm 1 shows the main process of labeling a trajectory with RM states. The first state of the trajectory s_0 is labeled with u_I (line 1). Then, we get the set of events ϕ_t and ϕ_{t+1} that s_t and s_{t+1} have completed by feature detector Φ (line 3). Let p_t be the set of events that are in ϕ_{t+1} but not in ϕ_t (line 4), which represents the set of events occurred in transition (s_t, a_t, s_{t+1}), and all events in p_t are passed to δ_u to get the next RM state (line 7).

Example 3.1. Consider the RM in Figure 2 and a trajectory with length $T = 200$. Assuming that the trajectory completes the event e_2 at step $t = 50$, cancels the e_2 at step $t = 60$, completes e_2 again at step $t = 70$ and completes e_3 at step $t = 120$. No events occur after step $t = 120$. This is a typical sub-optimal trajectory (In the Franka Kitchen domain, the robot may open the microwave, close the microwave, and open the microwave again).

Algorithm 1: Labeling a Trajectory with RM States

```
Input: event-ordered RM $\mathcal{R}^{eo} = \langle U, u_I, F, \Sigma', \delta_u, \delta_r \rangle$, feature
       detector $\Phi$, and trajectory $\{(s_t, a_t, s_{t+1})\}_{t=0}^{T-1}$.
1 $\tau \leftarrow \emptyset$, $u_0 \leftarrow u_I$; // Initialization.
2 for $t \leftarrow 0$ to $T-1$ do
      // Get events of states $s_t, s_{t+1}$ by detector $\Phi$.
3     $\phi_t \leftarrow \Phi(s_t)$, $\phi_{t+1} \leftarrow \Phi(s_{t+1})$;
      // Get new events in transition $(s_t, a_t, s_{t+1})$.
4     $p_t \leftarrow \phi_{t+1} - \phi_t = \{e | e \in \phi_{t+1} \wedge e \notin \phi_t\}$;
5     $u_{t+1} \leftarrow u_t$; // Copy the current RM state.
6     foreach $e$ in $p_t$ do
          // Update the next RM state with event $e$.
7         $u_{t+1} = \delta_u(u_{t+1}, e)$;
8     $\tau \leftarrow \tau \cup \{(s_t, u_t, a_t, s_{t+1}, u_{t+1})\}$;
9 return $\tau$; // A labeled trajectory with RM states.
```

As shown in Figure 3, the feature detector has the outputs: $\phi_t = \emptyset$ at steps $0 \leq t \leq 50$ and $61 \leq t \leq 70$, $\phi_t = \{e_2\}$ at steps $51 \leq t \leq 60$ and $71 \leq t \leq 120$, and $\phi_t = \{e_2, e_3\}$ at steps $121 \leq t \leq 200$. Then, the events that happen in transition (s_t, a_t, s_{t+1}) will be $p_t = \{e_2\}$ at steps $t = 50, 70$, $p_t = \{e_3\}$ at step $t = 120$ and $p_t = \emptyset$ for other steps. Finally, according to the transition function δ_u, the RM states will be $u_t = (e_0, e_0, e_0)$ at steps $0 \leq t \leq 50$, $u_t = (e_2, e_0, e_0)$ at steps $51 \leq t \leq 120$ and $u_t = (e_2, e_3, e_0)$ at steps $121 \leq t \leq 200$.

3.3 Splitting and Grouping Trajectories

After the offline dataset has been labeled with RM states, we can split the trajectories into sub-trajectories that complete one of the events. We define *event transitions* as the transitions where its RM states are different. In other words, a new event occurs in an event transition. Specifically, a transition (s, u, a, s', u') that satisfies $u \neq u'$ is a event transition. Given this, a whole trajectory can be split by these event transitions into sub-trajectories. In more details, we split all trajectories $\mathcal{T}$ into $\mathcal{T}^{\mathcal{R}} = \{\mathcal{T}^{u,u'}\}$, where $u, u' \in U$ and $u \neq u'$. Here, $\mathcal{T}^{u,u'}$ is the set of sub-trajectories that end with an event transition (s, u, a, s', u') and start from the first transition of a trajectory or the next transition of previous event transition. Notice that sub-trajectories that do not end with an event transition are abandoned. These sub-trajectories that do not trigger any event are considered less informative than those in $\mathcal{T}^{\mathcal{R}}$ for the task.

Example 3.2. Following the trajectory in Example 3.1, the event transitions are $\{(s_t, u_t, a_t, s_{t+1}, u_{t+1})\}$ with $t = 50, 120$, when the events e_2,e_3 first occur in the trajectory. Hence, the trajectory will be split into 3 sub-trajectories as: $\tau_1 = \tau^{0:50}$, $\tau_2 = \tau^{51:120}$, $\tau_3 = \tau^{121:200}$. Here, $\tau^{t_1:t_2} = \{(s_t, u_t, a_t, s_{t+1}, u_{t+1})\}_{t=t_1}^{t_2}$. After that, τ_1 will be added to the set $\mathcal{T}^{(e_0,e_0,e_0),(e_2,e_0,e_0)}$, τ_2 will be added to the set $\mathcal{T}^{(e_2,e_0,e_0),(e_2,e_3,e_0)}$, and τ_3 will be abandoned.

Since we have split the trajectories, the sub-trajectories can then be grouped by the event they end with and start from. Let $l : U \longrightarrow \Sigma'$ be a function that maps from a RM state into the latest event that occurs. For example, $l((e_2, e_1, e_0)) = e_1$. In particular, $l(u_I) = e_0$. All the sub-trajectories are grouped into $\{\mathcal{T}^{e_i \to e_j} | e_i \in \Sigma', e_j \in \Sigma\}$, where $\mathcal{T}^{e_i \to e_j}$ is defined as:

$$\mathcal{T}^{e_i \to e_j} = \bigcup \{\mathcal{T}^{u,u'} | l(u) = e_i, l(u') = e_j\} \tag{5}$$

For example, in the RM shown in Figure 2, the set $\mathcal{T}^{e_1 \to e_2}$ is the union of sets $\mathcal{T}^{(e_1,e_0,e_0),(e_1,e_2,e_0)}$ and $\mathcal{T}^{(e_3,e_1,e_0),(e_3,e_1,e_2)}$.

The sub-trajectories in the same group $\mathcal{T}^{e \to e'}$ are similar. All of them are ended with the occurrence of event e' and start after the occurrence of event e ($e \neq e_0$) or from the initial state of environment ($e = e_0$). For instance, sub-trajectory τ_2 in Example 3.2 is grouped into $\mathcal{T}^{e_2 \to e_3}$, which means that τ_2 completes the event e_3 at the end ($t = 120$) and its start transition ($t = 51$ is just the next transition of τ_1's end ($t = 50$), when the event e_2 occurs.

3.4 Refining Datasets with Reward Machines

3.4.1 Ordering Events by Trajectory Numbers. Now, we have the sub-trajectories grouped by the events $\Sigma = \{e_i\}_{i=1}^n$. We need to determine the order of the events. For some tasks, in order to complete a task, the events must occur in some special order. For example, the robot must open the door of a microwave before putting foods in it. In this case, we can eliminate the orders violating the requirements. If the tasks are indifferent of the event orders, we still need to decide one order for the events. Intuitively, this will make the policy training easier. Eventually, the agent will converge to a policy learning from the dataset, which completes the task in some order. Therefore, the event order is determined by the dataset. Given an order $(e^1, e^2, \ldots, e^n)$, we define the score of the order as $\prod_{i=1}^n |\mathcal{T}^{e^{i-1} \to e^i}|$, where $e^0 = e_0$ and $|\mathcal{T}|$ denotes the size of set $\mathcal{T}$. For example, in the RM shown in Figure 2, the score of the order (e_3, e_1, e_2) is $|\mathcal{T}^{e_0 \to e_3}| \times |\mathcal{T}^{e_3 \to e_1}| \times |\mathcal{T}^{e_1 \to e_2}|$. Intuitively, the score reflects how many trajectories that can be used to training the policy. Here, we select the order with highest score as the *best order* used in the next step.

3.4.2 Splicing Sub-trajectories with Ordered Events. Here, we use the best order $(e^1, e^2, \ldots, e^n)$ to splice the sub-trajectories and generate a new dataset as follow:

$$D = \bigcup_{i=1}^{n} \mathcal{T}^{e^{i-1} \to e^i} \tag{6}$$

where $e^0 = e_0$. Specifically, the new dataset is the union of the sets of sub-trajectories grouped by the events in the best order. For example, given the best order (e_3, e_1, e_2), the new dataset generated by (e_3, e_1, e_2) is $D = \mathcal{T}^{e_0 \to e_3} \cup \mathcal{T}^{e_3 \to e_1} \cup \mathcal{T}^{e_1 \to e_2}$. Since every trajectory in D associates with an event, this dataset is more suitable for offline RL because the events offer a guidance for completing the task. In contrast, the original dataset may contain many noisy and sub-optimal trajectories.

For sub-trajectories in the same group, we prefer the ones with shorter length because they can do the task more quickly. For example, in group $\mathcal{T}^{e_2 \to e_3}$, the trajectory τ_A of length 30 is better than the trajectory τ_B of length 50, since the event e_3 can be completed in 30 steps from e_2 in τ_A, and τ_B will need more steps.

In more details, for every group, we sort the trajectories from the short one to the long one by its length, and the top α trajectories are selected for splicing, where $0 < \alpha \leq 1$ is a hyper-parameter. For example, if $\alpha = 0.5$, the top 50% trajectories will be chosen. With fewer trajectories, the quality of the new dataset is improved.

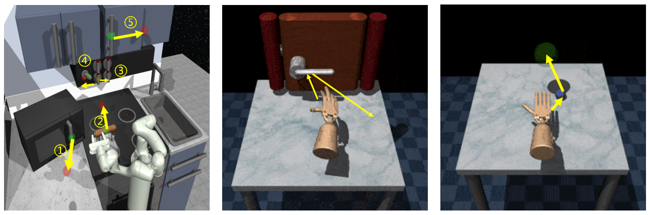
Algorithm 2: Relabeling with Counterfactual Experiences

```
Input: event-ordered RM $\mathcal{R}^{eo} = \langle U, u_I, F, \Sigma', \delta_u, \delta_r \rangle$, feature
       detector $\Phi$, the best event order $(e^1, e^2, \ldots, e^n)$, and
       the new dataset $D = \bigcup_{i=1}^n \mathcal{T}^{e^{i-1} \to e^i}$ where $e^0 = e_0$.
1  $D_C \leftarrow \emptyset, u_C \leftarrow u_I$; // Initialization.
2  for $i \leftarrow 1$ to $n$ do // Each event in the best order.
3      $\mathcal{T}_C \leftarrow \emptyset$;
4      $u'_C \leftarrow \delta_u(u_C, e^i)$; // Get next RM state.
5      $\mathcal{T} \leftarrow \mathcal{T}^{e^{i-1} \to e^i}$; // Get set of sub-trajectories.
6      foreach $\tau \in \mathcal{T}$ do /* $\tau = \{(s_t, u_t, a_t, s_{t+1}, u_{t+1})\}_{t=0}^{T-1}$
        denotes a sub-trajectory of length $T = |\tau|$. */
7          $\tau_C \leftarrow \emptyset$;
8          for $t \leftarrow 0$ to $T - 1$ do /* Relabel each
            transition of $\tau$ with RM states $u_C, u'_C$.   */
9              if $t = T - 1$ then // Last transition.
10                 $\tau_C \leftarrow \tau_C \cup \{(s_t, u_C, a_t, s_{t+1}, u'_C)\}$;
11             else
12                 $\tau_C \leftarrow \tau_C \cup \{(s_t, u_C, a_t, s_{t+1}, u_C)\}$;
13         $\mathcal{T}_C \leftarrow \mathcal{T}_C \cup \tau_C$;
14     $D_C \leftarrow D_C \cup \mathcal{T}_C$;
15     $u_C \leftarrow u'_C$; // Advance to next RM state.
16 return $D_C$; // Counterfactual trajectories.
```

3.4.3 Relabeling Datasets with Counterfactual Experiences. After grouping, the trajectories $\mathcal{T}^{e \to e'}$ may be labeled with different RM states. For example, in the RM shown in Figure 2, the set $\mathcal{T}^{e_1 \to e_2}$ is the union of sets $\mathcal{T}^{(e_1,e_0,e_0),(e_1,e_2,e_0)}$ and $\mathcal{T}^{(e_3,e_1,e_0),(e_3,e_1,e_2)}$. Assuming that the best order is $(e^1, e^2, e^3) = (e_3, e_1, e_2)$, the agent will learn a policy to complete the task following the order. Specifically, the order of RM states will be (u_0, u_1, u_2, u_3), where $u_0 = u_I = (e_0, e_0, e_0)$, $u_1 = (e_3, e_0, e_0)$, $u_2 = (e_3, e_1, e_0)$ and $u_3 = (e_3, e_1, e_2)$. After agent complete e_3, e_1 in order, the RM state will be u_2, and it needs to complete e_2. The agent learns the policy of completing e_2 from trajectories set $\mathcal{T}^{e_1 \to e_2}$ during the training. However, it can only do this from the sub-set $\mathcal{T}^{(e_3,e_1,e_0),(e_3,e_1,e_2)}$, not from $\mathcal{T}^{(e_1,e_0,e_0),(e_1,e_2,e_0)}$ when completing e_2. This is because its current RM state is $u_2 = (e_3, e_1, e_0) \neq (e_1, e_0, e_0)$. In this case, $\mathcal{T}^{(e_1,e_0,e_0),(e_1,e_2,e_0)}$ is not beneficial. Especially, if $\mathcal{T}^{(e_3,e_1,e_0),(e_3,e_1,e_2)}$ is empty, the agent cannot learn the policy of e_2.

To make use of all trajectories in $\mathcal{T}^{e \to e'}$, it is necessary to relabel the ones mentioned above and generate counterfactual trajectories, inspired by previous work on RM [8] [9]. For example, in $\mathcal{T}^{e_1 \to e_2}$, if all trajectories are labeled with u_2, agent can benefit from all trajectories in $\mathcal{T}^{e_1 \to e_2}$, though some of them may come from $\mathcal{T}^{(e_1,e_0,e_0),(e_1,e_2,e_0)}$. Hence, we generate counterfactual trajectories for groups selected according to the order of events.

The main process of generating counterfactual trajectories in the best event order is outlined in Algorithm 2. Given the best event order $(e^1, e^2, \ldots, e^n)$ and the newly generated dataset D, we augment the dataset by relabeling the RM states with the ones according to the best event order. Example 3.3 illustrates how we generate the counterfactual trajectories.



(a) kitchen (b) door (c) relocate

Figure 4: Simulated environments used in our experiments.

Example 3.3. Assuming that the best event order is $(e^1, e^2, e^3) = (e_1, e_2, e_3)$ in the RM shown in Figure 2. Starting with $u_0 = u_I$, the corresponding order of RM states is (u_0, u_1, u_2, u_3) where $u_1 = (e_1, e_0, e_0)$, $u_2 = (e_1, e_2, e_0)$ and $u_3 = (e_1, e_2, e_3)$. According to lines 1, 4 and 15 of Algorithm 2, u_C and u'_C will be u_{i-1} and u_i at i-th loop in line 2, where $i = 1, 2, 3$. In the i-th loop, the set of trajectories $\mathcal{T}^{e^{i-1} \to e^i}$ are chosen (line 5), and all trajectories in the set will be relabeled with $u_C = u_{i-1}$ and $u'_C = u_i$ to generate counterfactual trajectories. For example, consider the trajectory τ_2 in Example 3.2, which belongs to the set $\mathcal{T}^{(e_2,e_0,e_0),(e_2,e_3,e_0)}$. Since $\mathcal{T}^{(e_2,e_0,e_0),(e_2,e_3,e_0)} \subseteq \mathcal{T}^{e_2 \to e_3}$, and $e^2 = e_2, e^3 = e_3$, it will be chosen at $i = 3$. $u_C = u_2$ and $u'_C = u_3$ at $i = 3$. The original RM state of τ_2 is $u_t = (e_2, e_0, e_0), 0 \le t \le T - 1$ and $u_T = (e_2, e_3, e_0)$. When τ_2 is chosen in line 6, it will replace (e_2, e_0, e_0) with u_2 and (e_2, e_3, e_0) with u_3 in lines 8-12.

3.5 Learning Policy with Refined Datasets

Given the refined datasets generated by our method, we can generally use any offline RL methods to learn the policy. In our experiments, we use IQL [10], which is currently the leading offline RL algorithm. In our implementation, we concatenate the environment state and the corresponding RM states as the input states. During evaluation stage, the feature detector Φ is used to retrieve the events. Specifically, given the state s and RM state u, the agent gets the next state s' by taking action a. Then the events from the transition are computed by $p = \Phi(s') - \Phi(s)$ and all the events in p are passed into the RM to obtain the next RM state u'.

To avoid overfitting with small datasets, we apply weight decay to regularize the policy and Q-value networks. Specifically, the loss functions in Equations 2 and 4 are simply converted to:

$$L(\theta) = \mathbb{E}_{(s,a,r,s') \sim D} \left[(r + \gamma V_\psi(s') - Q_\theta(s,a))^2 \right] + \lambda_Q ||\theta||^2 \quad (7)$$

$$L(\phi) = \mathbb{E}_{(s,a) \sim D} \left[\exp(\beta(Q_{\hat{\theta}}(s,a) - V_\psi(s))) \log \pi_\phi(a|s) \right] + \lambda_\pi ||\phi||^2 \quad (8)$$

where λ_Q and λ_π are the hyper-parameters.

4 EXPERIMENTS

In this section, we empirically evaluate of our method in two benchmark domains: 1) the Franka Kitchen domain [7] and 2) the Adroit domain [20], whose simulated environments are shown in Figure 4. Specifically, we test our method on the D4RL benchmark [5]. The datasets are sub-optimal and task-agnostic, which are challenging and commonly used in offline RL. Additionally, we analyze how main procedures of our method refine the datasets.

Table 1: The average reward for all the tested datasets and methods (Best values in bold).

Domains	Datasets	BC	BCQ	IQL	CQL	Ours
Kitchen	kitchen-partial-v0	1.31	0.46	1.60	0.92	**3.24**
	kitchen-mixed-v0	1.61	0.62	1.68	0.53	**3.03**
Adroit	door-human-v0	-20.4	-54.8	74.8	75.8	**439.5**
	door-cloned-v0	-56.1	-57.5	20.1	-56.5	**415.9**
	relocate-human-v0	184.2	-25.1	288.6	78.1	**633.3**
	relocate-cloned-v0	-51.6	-56.0	-52.4	-30.1	**492.4**

We compare our method with the following baselines: Behavior Cloning (BC), BCQ [6], CQL [12] and IQL [10]. For the baselines, we use the authors' open-sourced implementation with the recommended hyper-parameters reported in their papers. We train the policy for 0.5M gradient steps. The policy is evaluated every 2000 steps over 10 episodes. We use the average reward from the original environments in every episode to report the performance of each method. Results are averaged over three random seeds.

4.1 Franka Kitchen Domain

4.1.1 Settings. The Franka Kitchen domain involves controlling a 9-DoF Franka robot in a kitchen environment to manipulate multiple objects (e.g., microwave, kettle, etc.). It is a multi-task environment. An example task is to complete the following sub-tasks: 1) open the microwave, 2) slide the cabinet door, 3) place the kettle on the top burner, and 4) turn the overhead light on. The events for building the RM can be easily defined as the completion of every aforementioned sub-task.

We test on two D4RL datasets: 1) *kitchen-partial-v0* and 2) *kitchen-mixed-v0*. Both of them consist of undirected data, where the robot performs sub-tasks that are not necessarily related to the goal task. In *kitchen-partial-v0* dataset, most of the trajectories are sub-optimal and less than 3% of them complete the whole task. In *kitchen-mixed-v0* dataset, none of the trajectories complete the whole task [5], which is more challenging for the agent to learn.

In the dataset, states from the environment are 60-dimensional, which consist of a 9-dimensional vector that describes the 9-DoF robot's joint velocities, a 21-dimensional vector that represents the poses of objects, and the remain for a fixed goal related to the task of the environment. The original environment will give the agent a reward of +1 when any sub-task is completed for the first time and every task consists of four sub-tasks.

4.1.2 Results. As shown in Figures 5a and 5b and Table 1, our method outperforms all the baselines by a large margin. In fact, only our method is able to successfully complete the whole task. It can be observed that our method can complete the whole task in a particular order, while the others failed. For example, in the *kitchen-partial-v0*, the agent with our method first opened the microwave, then placed the kettle, turned the light on, and finally slid the cabinet door. With other methods, the agent tended to place the kettle first, since the kettle is closer to the initial position of the robot. Unfortunately, there are no trajectories opening the microwave after placing the kettle in the dataset. As a result, the agent with other methods could not open the microwave later and therefore failed to complete the whole task. In the *kitchen-mixed-v0*, though there are no trajectories complete the whole task, our method is still able to complete 3 sub-tasks at least and sometimes complete the whole task. These results show that our method with dataset refinement and counterfactual reasoning is beneficial for offline RL.

4.2 Adroit Domain

4.2.1 Settings. The Adroit domain involves controlling a 24-DoF simulated hand tasked with opening a door or picking up and moving a ball. We test on two datasets: 1) *human*: 25 demonstration trajectories from a human, 2) *cloned*: trajectories collected by an imitation policy trained on the demonstrations.

For the *door* task, the agent needs to open the door. To build the RM, we define the set of events as e_1: $d < 0.1$, e_2: $\theta > 0.2$, e_3: $\theta > 1.0$ and e_4: $\theta > 1.35$, where d is the distance between the palm and the door handle and θ is the opening angle of the door. Each episode lasts 200 time steps.

For the *relocate* task, the agent needs to pick up a ball and place it to the target position. To build the RM, we define the set of events as e_1: $d_1 < 0.1$, e_2: $d_2 < 0.1$ and e_3: $d_2 < 0.05$ where d_1 is the distance between the palm and the ball and d_2 is the distance between the ball and the target. The initial position of the ball and target is random and each episode lasts 600 time steps.

4.2.2 Results. As shown in Figure 5 and Table 1, our method outperforms all the baselines. In the *cloned* dataset, while most of the baselines failed to reach the door or ball, our method successfully completed the whole task. Most of the trajectories in the *cloned* dataset are task-agnostic and noisy, which makes other methods difficult to learn a policy to complete the task. In our method, the noisy trajectories are removed during trajectories splitting with RM states. This can significantly improve the performance. Figures 5c and 5e show that our method can complete task faster than the others. Although all of the trajectories in the *human* dataset complete the whole task, our method only select the shorter trajectories to learn the policy. These results show that our method can generate trajectories of higher quality, which is useful for offline RL.

4.3 Analysis on Main Procedures

Here, we analyze how each procedure of our method affects the size of the datasets, which is measured by the number of transitions in datasets. In our method, there are three procedures that will reduce the size of the datasets. Table 2 show the size of the datasets after each procedure for all the test domains. Note that the original dataset consists of sub-optimal and task-agnostic trajectories.

4.3.1 Splitting and Grouping. In this procedure, we will split all trajectories into sub-trajectories. Note that sub-trajectories that do not occur any event will be abandoned. Intuitively, those abandoned sub-trajectories are task-agnostic and noisy for policy learning. As shown in Table 2, this procedure significantly reduces the *door-cloned* and *relocate-cloned* datasets, which abandon more than 60% transitions in the original datasets. We observed that most of the abandoned trajectories do not even reach the door or the ball, which behave randomly and will have negative impact on the performance of offline RL. For the two *kitchen* datasets, many trajectories complete only one or two of the sub-tasks. Here, sub-trajectories completing irrelevant sub-tasks are abandoned. It can be seen that this procedure can remove noisy transitions that are task-agnostic. This make the datasets more focusing on the task.

4.3.2 Splicing with Ordered Events. In this procedure, we will select sub-trajectories that are in the best order extracted from datasets. Note that trajectories that are out of order are abandoned. This procedure affects the two *kitchen* datasets at most. Though all trajectories after splitting complete the required sub-tasks, some of them are not useful for policy learning. For example, by following the best order for *kitchen-partial*, the agent should open the microwave in the beginning. However, some trajectories place the kettle, turn the light on or slide the cabinet first. Those tasks should be completed after opening the microwave according to the best order. In the datasets, none of trajectories open the microwave after completing other sub-tasks and all trajectories with opening the microwave start from the initial position. Hence, the agent may fail to learn a policy to complete the whole task if it does not open the microwave first. Since the *door* and *relocate* have only a single event order, they are not affected by this procedure.

4.3.3 Selecting Shorter Trajectories. In this procedure, we will select the shorter trajectories in a group, which is controlled by a hyper-parameter α. In our experiments, we set $\alpha = 0.5$ for *kitchen-partial* and the two *door* datasets, $\alpha = 0.9$ for *kitchen-mixed* and $\alpha = 1$ for the two *relocate* datasets. Although all trajectories in the group are useful for completing the task, some of them are sub-optimal. For example, in the *door* datasets, some of them pull the door very slowly. By removing them, we can make the agent learn a policy of opening door more quickly. Thus, this procedure can further optimize the datasets.

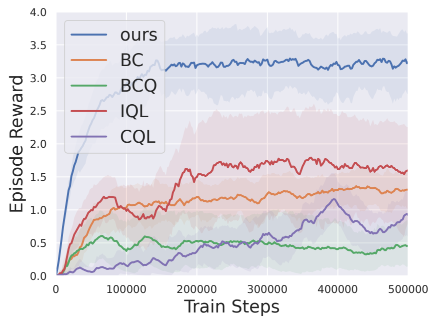
In summary, our results in Tables 1 and 2 show that our methods can achieve higher performance on smaller datasets (i.e., less is more), which confirm the effectiveness of our method.

5 RELATED WORK

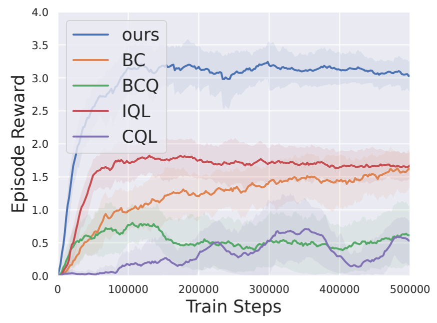
5.1 Offline Reinforcement Learning

Offline RL aims to address the problem of learning effective policies entirely from previously collected data collected by some behavior policies, without online interaction [6, 13]. The main challenge is the overestimation of value function of OOD actions, which leads to performance degrade [6]. To address this problem, methods like BCQ [6], BEAR [11] and BRAC [27] constrain the learned policy to the behavior policy used to collect the dataset. Other methods [12, 29] constrain the learned policy by making conservative estimates of value functions of OOD actions, for example, CQL [12] enforces a regularization constraint on the critic loss to penalize value function of OOD actions. A few methods avoid the problem by taking a single step of policy evaluation and policy improvement [2, 10], which never query value function of OOD actions. In practice, IQL has shown to be one of the most successful methods on D4RL [19].

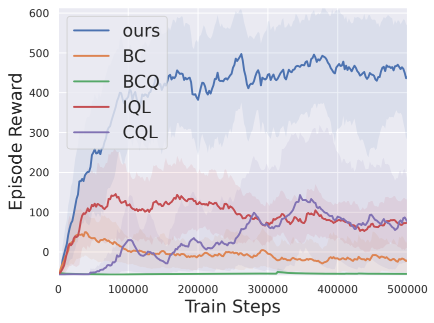
Another line of work tries to make better use of datasets. For instance, S4RL [23] uses data augmentations to improve the function approximation for Q-learning algorithms in offline RL. Methods like [1, 17, 18, 22] extract skills from offline datasets to accelerate offline [1, 22] or online [17, 18] RL to complete the task. In contrast to these works, we focus on refining dataset with prior knowledge to improve the performance of offline RL methods.



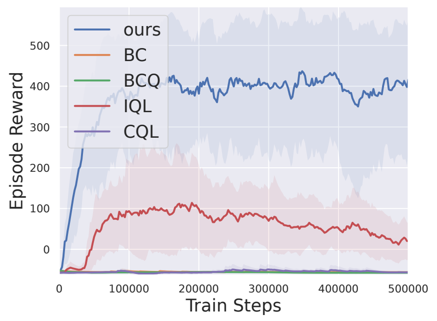
(a) kitchen-partial-v0



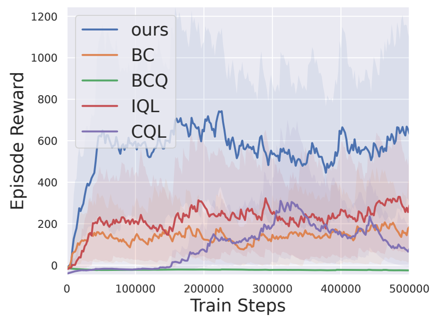
(b) kitchen-mixed-v0



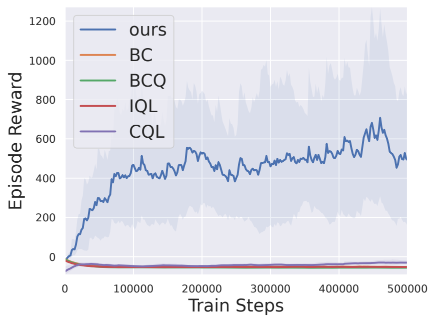
(c) door-human-v0



(d) door-cloned-v0



(e) relocate-human-v0



(f) relocate-cloned-v0

Figure 5: The learning curve of our method and the baselines in the Franka Kitchen and Adroit (door and relocate) domains.

Table 2: The number of transitions in all the tested datasets after each procedure.

Procedures	kitchen-partial	kitchen-mixed	door-human	door-cloned	relocate-human	relocate-cloned
Original Dataset	136345	136345	6704	995643	9917	996242
Splitting Traj.	98690	83792	5001	372086	7211	364207
Ordered Events	50114	49855	5001	372086	7211	364207
Shorter Traj.	22682	44298	2261	167397	7211	364207

5.2 Reward Machines

RM is a finite-state machine that encodes reward functions for MDPs [8]. RM is initially introduced in [8], in which Q-learning with RMs (QRM) was proposed. QRM is an algorithm that learns a set of Q-functions for all RM states and share experiences with all RM states. Later [9] proposed counterfactual experiences for RMs (CRM), which is another version of QRM that learns one Q-function which take RM states as one of the inputs and is more suitable when combining with deep networks. In contrast to them, we instead generate counterfactual experiences by replacing the RM states with particular ones, which is more suitable for offline RL.

Additionally, RM has been used for solving problems in robotics [3, 4, 21], multi-agent systems [15], lifelong RL and partial observability [25]. Some approaches [25, 26, 28] try to learn RMs from experience. Unlike these existing methods, we model human knowledge with a RM and show the benefits of RM for offline RL.

6 CONCLUSIONS

In this paper, we proposed a data refinement method to learn a policy for long-horizon tasks with sub-optimal and task-agnostic offline datasets. Using the RM, we label and split the trajectories and get the best order of the events of the task. Given the order, we select trajectories and generate counterfactual experiences. By doing so, offline RL methods can learn better policy from the refined datasets than the original ones. Our experiments on the D4RL benchmark demonstrate the efficiency of our method, which learn a good policy from relatively low-quality datasets. We show how the RMs can improve the performance of offline RL by decomposing the task and reduce the noise. In the future, we plan to extend our method to make better use of the datasets. For instance, the abandoned sub-trajectories in the grouping and splicing processes can be reused. This will be useful for domains where only small datasets are available or the process of data collection is costly.

ACKNOWLEDGMENTS

This work was supported in part by the Major Research Plan of the National Natural Science Foundation of China (Grant No. 92048301) and Anhui Provincial Natural Science Foundation (Grant No. 2208085MF172).

REFERENCES

[1] Anurag Ajay, Aviral Kumar, Pulkit Agrawal, Sergey Levine, and Ofir Nachum. 2020. Opal: Offline primitive discovery for accelerating offline reinforcement learning. *arXiv preprint arXiv:2010.13611* (2020).

[2] David Brandfonbrener, Will Whitney, Rajesh Ranganath, and Joan Bruna. 2021. Offline rl without off-policy evaluation. *Advances in Neural Information Processing Systems* 34 (2021), 4933–4946.

[3] Alberto Camacho, Jacob Varley, Andy Zeng, Deepali Jain, Atil Iscen, and Dmitry Kalashnikov. 2021. Reward machines for vision-based robotic manipulation. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 14284–14290.

[4] David DeFazio and Shiqi Zhang. 2021. Learning quadruped locomotion policies with reward machines. *arXiv preprint arXiv:2107.10969* (2021).

[5] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. 2020. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219* (2020).

[6] Scott Fujimoto, David Meger, and Doina Precup. 2019. Off-policy deep reinforcement learning without exploration. In *International conference on machine learning*. PMLR, 2052–2062.

[7] Abhishek Gupta, Vikash Kumar, Corey Lynch, Sergey Levine, and Karol Hausman. 2019. Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning. *arXiv preprint arXiv:1910.11956* (2019).

[8] Rodrigo Toro Icarte, Toryn Klassen, Richard Valenzano, and Sheila McIlraith. 2018. Using reward machines for high-level task specification and decomposition in reinforcement learning. In *International Conference on Machine Learning*. PMLR, 2107–2116.

[9] Rodrigo Toro Icarte, Toryn Q Klassen, Richard Valenzano, and Sheila A McIlraith. 2022. Reward machines: Exploiting reward function structure in reinforcement learning. *Journal of Artificial Intelligence Research* 73 (2022), 173–208.

[10] Ilya Kostrikov, Ashvin Nair, and Sergey Levine. 2021. Offline reinforcement learning with implicit q-learning. *arXiv preprint arXiv:2110.06169* (2021).

[11] Aviral Kumar, Justin Fu, Matthew Soh, George Tucker, and Sergey Levine. 2019. Stabilizing off-policy q-learning via bootstrapping error reduction. *Advances in Neural Information Processing Systems* 32 (2019).

[12] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. 2020. Conservative q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems* 33 (2020), 1179–1191.

[13] Sascha Lange, Thomas Gabel, and Martin Riedmiller. 2012. Batch reinforcement learning. In *Reinforcement learning*. Springer, 45–73.

[14] Jinming Ma and Feng Wu. 2023. Learning to Coordinate from Offline Datasets with Uncoordinated Behavior Policies. In *Proceedings of the 22nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2023)*. London, United Kingdom.

[15] Cyrus Neary, Zhe Xu, Bo Wu, and Ufuk Topcu. 2020. Reward machines for cooperative multi-agent reinforcement learning. *arXiv preprint arXiv:2007.01962* (2020).

[16] Xue Bin Peng, Aviral Kumar, Grace Zhang, and Sergey Levine. 2019. Advantage-weighted regression: Simple and scalable off-policy reinforcement learning. *arXiv preprint arXiv:1910.00177* (2019).

[17] Karl Pertsch, Youngwoon Lee, and Joseph Lim. 2021. Accelerating reinforcement learning with learned skill priors. In *Conference on robot learning*. PMLR, 188–204.

[18] Karl Pertsch, Youngwoon Lee, Yue Wu, and Joseph J Lim. 2021. Guided reinforcement learning with learned skills. *arXiv preprint arXiv:2107.10253* (2021).

[19] Rafael Figueiredo Prudencio, Marcos ROA Maximo, and Esther Luna Colombini. 2022. A Survey on Offline Reinforcement Learning: Taxonomy, Review, and Open Problems. *arXiv preprint arXiv:2203.01387* (2022).

[20] Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. 2017. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *arXiv preprint arXiv:1709.10087* (2017).

[21] Ankit Shah, Shen Li, and Julie Shah. 2020. Planning with uncertain specifications (puns). *IEEE Robotics and Automation Letters* 5, 2 (2020), 3414–3421.

[22] Noah Y Siegel, Jost Tobias Springenberg, Felix Berkenkamp, Abbas Abdolmaleki, Michael Neunert, Thomas Lampe, Roland Hafner, Nicolas Heess, and Martin Riedmiller. 2020. Keep doing what worked: Behavioral modelling priors for offline reinforcement learning. *arXiv preprint arXiv:2002.08396* (2020).

[23] Samarth Sinha, Ajay Mandlekar, and Animesh Garg. 2022. S4RL: Surprisingly simple self-supervision for offline reinforcement learning in robotics. In *Conference on Robot Learning*. PMLR, 907–917.

[24] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction.* MIT press.

[25] Rodrigo Toro Icarte, Ethan Waldie, Toryn Klassen, Rick Valenzano, Margarita Castro, and Sheila McIlraith. 2019. Learning reward machines for partially observable reinforcement learning. *Advances in neural information processing systems* 32 (2019).

[26] Alvaro Velasquez, Andre Beckus, Taylor Dohmen, Ashutosh Trivedi, Noah Topper, and George Atia. 2021. Learning probabilistic reward machines from non-Markovian stochastic reward processes. *arXiv preprint arXiv:2107.04633* (2021).

[27] Yifan Wu, George Tucker, and Ofir Nachum. 2019. Behavior regularized offline reinforcement learning. *arXiv preprint arXiv:1911.11361* (2019).

[28] Zhe Xu, Ivan Gavran, Yousef Ahmad, Rupak Majumdar, Daniel Neider, Ufuk Topcu, and Bo Wu. 2020. Joint inference of reward machines and policies for reinforcement learning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, Vol. 30. 590–598.

[29] Tianhe Yu, Aviral Kumar, Rafael Rafailov, Aravind Rajeswaran, Sergey Levine, and Chelsea Finn. 2021. Combo: Conservative offline model-based policy optimization. *Advances in neural information processing systems* 34 (2021), 28954–28967.