A The MDP Environment of CRS

The MDP environment describes the current state, state transition, and the possible actions that the agent could take.

A.1 State

The state $s_t \in \mathcal{S}$ contains all the conversation history till timestep t. Given a user u, the state is defined as,

$$s_t = [\mathcal{P}_t^+, \mathcal{P}_t^-, \mathcal{V}_t^-], \tag{1}$$

where (1) $\mathcal{P}_t^+$ is the set of attributes accepted by the user during the conversation; (2) $\mathcal{P}_t^-$ is the set of attributes dismissed by the user during the conversation; (3) $\mathcal{V}_t^-$ is the set of items rejected by the user. In addition, we also add the set of candidate items $\mathcal{V}_t^{cand}$ that satisfy all inquired attributes to enrich the state information. The state s_t is input into a state encoder. The resulted state embedding will be concatenated with the user embedding e_u, which is pre-trained from given historical interaction data.

A.2 Action

According to the state s_t, the agent takes an action $a_t \in \mathcal{A}$, where a_t can be an item v_t from the candidate item set $\mathcal{V}_t^{cand}$ to make a recommendation or an attribute p_t from the candidate attribute set $\mathcal{P}_t^{cand}$ to ask attributes. Following the path reasoning approach [Lei et al., 2020], we identify the candidate item and candidate attribute as follows,

$$\mathcal{V}_t^{cand} = \mathcal{V}_{\mathcal{P}_t^+} \setminus \mathcal{V}_t^-, \ \mathcal{P}_t^{cand} = \mathcal{P}_{\mathcal{V}_t^{cand}} \setminus (\mathcal{P}_t^+ \cup \mathcal{P}_t^-), \tag{2}$$

where $\mathcal{V}_{\mathcal{P}_t^+}$ is the set of items satisfying all $\mathcal{P}_t^+$, and $\mathcal{P}_{\mathcal{V}_t^{cand}}$ is the set of attributes belonging to at least one of the candidate items. Specifically, when an item v_t is selected by the policy, top-K items with highest ranking scores will be selected to form a recommendation list $\mathcal{V}_t^{rec}$. K is set to 10 in our experiments.

A.3 State Transition

The current state s_t will transit to the next state s_{t+1} when the user responds to the action a_t. Specifically, if the agent asks an attribute p_t and the user accepts it, the next state s_{t+1} will be updated by $\mathcal{P}_{t+1}^+ = \mathcal{P}_t^+ \cup \{p_t\}$. Conversely, if the user rejects the attribute p_t, s_{t+1} will be updated by $\mathcal{P}_{t+1}^- = \mathcal{P}_t^- \cup \{p_t\}$. Also, if the user rejects the recommendation $\mathcal{V}_t^{rec}$, we will update $\mathcal{V}_{t+1}^- = \mathcal{V}_t^- \cup \mathcal{V}_t^{rec}$.

A.4 Action Selection Strategy

A large action search space will affect the performance of policy learning. We follow [Deng et al., 2021] to use the following two heuristics to pre-select the actions to facilitate policy learning.

Preference-based Item Selection. We select top-K_v candidate items from $\mathcal{V}_t^{cand}$ into the candidate action space $\mathcal{A}_t$ at each timestep t. The ranking score of an item v is given by

$$w_t(v) = e_u^\top e_v + \sum_{p \in \mathcal{P}_t^+} e_v^\top e_p - \sum_{p \in \mathcal{P}_t^- \cap \mathcal{P}_v} e_v^\top e_p. \tag{3}$$

where $\{e_u, e_v, e_p\}$ represent the pre-trained user, item and attribute embeddings from given historical interaction data. We require $K_v \geq K$. In our experiments, both of them are set to 10.

Weighted Entropy-based Attribute Selection. Whereas for candidate attributes to be asked, the expected one is supposed to be able to not only better eliminate the uncertainty of candidate items, but also encode the user preference. Inspired by [Lei et al., 2020], we adopt weighted entropy as the criteria to prune candidate attributes,

$$w_t(p) = -\text{prob}(p_t) \cdot \log(\text{prob}(p_t)), \tag{4}$$

$$\text{prob}(p_t) = \frac{\sum_{v \in \mathcal{V}_t^{cand} \cap \mathcal{V}_p} w_t(v)}{\sum_{v \in \mathcal{V}_t^{cand}} w_t(v)},$$

where $\mathcal{V}_p$ denotes the items that have the attribute p. Similar to item selection, we select top-K_p candidate attributes from $\mathcal{P}_t^{cand}$ into $\mathcal{A}_t$ based on the weighted entropy score, which is set to 10 in our experiments.

B More Experiment Results

A recent work [Hu et al., 2022] proposes a heuristic to identify the best action at each timestep. Specifically, they use a simple two-action policy, which only decides whether to ask an attribute question or make a recommendation. When the policy decides to ask a question, the attribute with the largest score given by Eq.(4) will be selected. When the policy decides to recommend, the items will be ranked according to Eq.(3) and the policy recommends the top-K items. The preferred action at each timestep is decided by Algorithm 1 based on the authors' proposed heuristics. Then they devise the binary reward signal based on this preferred action, and use the learned reward function for policy learning. The resulted method is named as **CRIF**.

Algorithm 1: Action Judgement of Pre-defined Rules

```
Input: The size of candidate item set in the last turn $l_b$, the rank of target item in the last turn $k_b$,
  the size of candidate item set after asking an attribute $l_a$, the rank of target item after asking an
  attribute $k_a$, the size of candidate item set after the recommendation $l_r$, and the rank of target
  item after the recommendation $k_r$;
Output: Action comparison result: asking an attribute is better or recommendation is better;
Target item rank margin of ask $k_a^+ = k_a - k_b$;
Candidate item length margin of ask $l_a^+ = l_a - l_b$;
Target item rank margin of recommendation $k_r^+ = k_r - k_b$;
Candidate item length margin of recommendation $l_r^+ = l_r - l_b$;
if $l_b \leq 50$ then
    if $l_b \leq 10$ then
        recommendation better; Exit.
    end
    else
        if $k_a^+ > k_r^+$ then
            ask better; Exit.
        end
        else
            recommendation better; Exit.
        end
    end
    else
        if $k_a^+ + 0.5 * l_a^+ > k_r^+ + 0.5 * l_r^+$ then
            ask better; Exit.
        end
        else
            recommendation better. Exit.
        end
    end
end
```

However, this heuristic abuses the design of user simulator that all attributes of the target item will be accepted. This however creates a form of information leakage and creates an unfair advantage over the baseline solutions that do not exploit this specific knowledge. But it is still interesting to understand whether our learned intrinsic rewards can augment this strong heuristic. Instead of learning a reward function to match the heuristic, we directly use rules given by Algorithm 1 to define the extrinsic rewards: we set an action's extrinsic reward to 1 if the select action is the same as the ground-truth action given by Algorithm 1; otherwise, 0. To make the problem more difficult, we set the maximum allowed turns to 10, and thus the policy needs to recommend successfully with fewer conversation turns. We first train a policy with policy gradient using the rule-based extrinsic reward, and then fine-tune the policy using CRSIRL.

Table 1: Results with rule-based extrinsic reward.

	LastFM			LastFM*			Yelp*		
	SR@10	AT	hDCG	SR@10	AT	hDCG	SR@10	AT	hDCG
CRIF	0.784	9.37	0.242	0.827	7.58	0.376	0.636	9.38	0.214
PG+Rules	0.840	8.89	0.261	**0.952**	**6.82**	0.415	0.723	8.95	0.242
CRSIRL+Rules	**0.862**	**8.84**	**0.274**	0.946	6.93	**0.421**	**0.756**	**8.81**	**0.248**

The results are presented in Table 1. Interestingly, directly optimizing extrinsic rewards in PG+Rules already outperformed CRIF, suggesting that the estimation errors in CRIF's reward function can lead to performance degeneration. The rule-based extrinsic reward is already dense and specifically designed for conversational recommendation, yet CRSIRL still enhances performance generally (as seen on LastFM and Yelp*). However, when the pre-defined rules fit the dataset well, such as on LastFM*, the learned intrinsic rewards could be less helpful. Nevertheless, Algorithm 1 still contains some manually defined parts, such as the weights of k^+ and l^+, which have to be finetuned in a per-dataset basis. CRSIRL, by learning intrinsic rewards directly from user interactions, offers an effective augmentation to these potentially flawed handcrafted rules.

Moreover, we should note that even though CRSIRL is built upon the unified policy proposed in [Deng et al., 2021], it is a general method can be easily applied to other existing RL-based CRS solutions. We additionally applied CRSIRL to SCPR [Lei et al., 2020], which reformulates the CRS problem as an interaction path reasoning process within a user-item-attribute graph. Specifically, they employ a two-action policy, which decides to ask a question or make a recommendation. The policy is optimized using the pre-defined reward described in Section 5.1. We use CRSIRL to fine-tune the policy obtained by SCPR. We present the results in Table 2. It shows CRSIRL can improve the performance significantly with the learned intrinsic rewards.

Table 2: Results of applying CRSIRL on SCPR.

	LastFM			LastFM*			Yelp*		
	SR@15	AT	hDCG	SR@15	AT	hDCG	SR@15	AT	hDCG
SCPR	0.465	12.86	0.139	0.709	8.43	0.317	0.489	12.62	0.159
CRSIRL+SCPR	**0.614**	**11.32**	**0.184**	**0.773**	**8.06**	**0.338**	**0.543**	**11.02**	**0.198**

References

Yang Deng, Yaliang Li, Fei Sun, Bolin Ding, and Wai Lam. Unified conversational recommendation policy learning via graph-based reinforcement learning. *arXiv preprint arXiv:2105.09710*, 2021.

Chenhao Hu, Shuhua Huang, Yansen Zhang, and Yubao Liu. Learning to infer user implicit preference in conversational recommendation. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 256–266, 2022.

Wenqiang Lei, Gangyi Zhang, Xiangnan He, Yisong Miao, Xiang Wang, Liang Chen, and Tat-Seng Chua. Interactive path reasoning on graph for conversational recommendation. In *Proceedings of the 26th ACM SIGKDD*, pages 2073–2083, 2020.