ReinForest: Multi-Domain Dialogue Management Using Hierarchical Policies and Knowledge Ontology

Tiancheng Zhao

CMU-LTI-16-006

Language Technologies Institute
School of Computer Science
Carnegie Mellon University
5000 Forbes Ave., Pittsburgh, PA 15213
www.lti.cs.cmu.

© April 4, 2016
ReinForest: Multi-Domain Dialogue Management Using Hierarchical Policies and Knowledge Ontology

Tiancheng Zhao
Language Technologies Institute
Carnegie Mellon University
April 4, 2016

1 Introduction

This report describes a dialog management framework that is designed to efficiently create multi-domain, mixed-initiative dialog system. We formalize the existing state-of-the-art plan-based RavenClaw dialog management framework [2] into a Semi-Markov Decision Process (SMDP), which finds its root in the hierarchical reinforcement learning (HRL) [10, 5] community. The proposed framework also incorporates a domain-dependent ontology for developers to rapidly encode domain knowledge. Therefore, the proposed model allows rapid extension to new domains and profits from dialog policy reusability.

2 Related Work

ReinForest is closely related to the RavenClaw [2] dialog management framework. RavenClaw is the state-of-art plan-based architecture that has been used to develop a large number of real-world dialog systems, including RoomLine, Let’s Go Public [9], LARRI [3], Team Talk [6], VERA etc. RavenClaw introduced the idea of task-independence of the dialog manager, which only requires developers to specify a domain-related dialog task tree. After the task tree is defined, a domain-independent dialog engine can execute the dialog tree and conducts error recovery based on a library of error-handling strategies. Although it is much easier to define domain related dialog task trees comparing to naive flow-chart approach [7], the dialog task tree can still explode to intractable size when dealing with multi-domains. Thus ReinForest takes one step further so that developers only need to specify a knowledge ontology for all domains. The generation and the execution of the dialog task trees are both domain independent.

Moreover, ReinForest formalizes the execution of the dialog task tree into a Semi Markov Decision Process (SMDP) framework. The theory of SMDPs provides the foundation for hierarchical reinforcement learning (HRL), which allows arbitrary structural knowledge to be encoded in the policies of an agent. Past work [5, 10, 1, 11] has shown that HRL can achieve better sample-efficiency and policy resuability compared to plain reinforcement learning algorithms. Therefore, unifying traditional plan-based dialog management with SMDPs opens up the possibilities of training plan-based dialog manager using principled machine learning algorithms from the HRL framework.

3 Interface

We first define the input and output API used by the ReinForest Dialog Manager (RM), which is outlined in Figure 1.

The input to ReinForest is a semantic frame from an external natural language understanding (NLU). The frame contains the original utterance along with the three annotations: entities, an intent and a domain. Assuming the user utterance is: "Recommend a restaurant for me in Pittsburgh", an example semantic frame input to ReinForest is as follows:
Figure 1: The input to ReinForest is a semantic frame that comes from the natural language understanding (NLU). The output of ReinForest is a list of dialog acts and content value pairs to the natural language generator (NLG).

<table>
<thead>
<tr>
<th>Domain</th>
<th>{Restaurant: 0.95; Hotel: 0.05}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intent</td>
<td>{Request: 0.9; Inform: 0.1; Confirm: 0.0}</td>
</tr>
<tr>
<td>Entities</td>
<td>{Type: Location; Value: Pittsburgh}</td>
</tr>
</tbody>
</table>

Given the user input annotated by the NLU, ReinForest will update its internal dialog state and generate the next system response. The response is in the format of a list of dialog acts and content value tuples, \( a_{sys} = [(DA_0, v_0), \ldots (DA_k, v_k)] \). The NLG is responsible for transforming \( a_{sys} \) to the natural language surface form and replying to the user. A valid ReinForest response given the utterance in the previous example is as follows, which should be transformed to: "I believe you said Pittsburgh. What kind food do you want?" by a simple NLG:

\[
  a_{sys} = [(CONFIRM, value = Pittsburgh), (ASK, value = food_type)]
\]  

(1)

4 System Architecture

The overall architecture of ReinForest is shown in Figure 2.

Figure 2: The architecture of ReinForest. ReinForest has a domain-independent dialog engine operating on a domain-dependent knowledge ontology.

Therefore, the core of ReinForest has two parts: knowledge ontology and dialog engine. The knowledge ontology is a domain-dependent knowledge graph where developers can quickly encode domain knowledge and relations between various concepts. Meanwhile, the dialog engine is a domain-independent execution mechanism that generates the next system response given the current dialog state. The following sections formally define these two components.

5 Knowledge Ontology

Concept: The knowledge Ontology can be thought of as the fuel for the ReinForest dialog engine. The basic unit of the ontology is a concept. The internal structure of a concept is outlined in Figure 3 and formally defined as:
Figure 3: A concept is the basic unit of knowledge ontology. A concept has an attribute map (a Hashmap) that stores attribute information. It is optional to have a set of subscribed entities and domains, linked to external databases and dependencies linked to other concepts (e.g. \( C_0 \) depends on \( C_1 \) to \( C_k \)).

1. An attribute map: essentially a hashmap that maps from a string key to an attribute (defined below).
2. A set of subscribed entities and domains: when the user input utterance contains the subscribed entities or domains, the dialog engine will trigger handling functions to perform updates on the concept.
3. A set of dependency concepts: a concept can depend on a set of other concepts before its value can be obtained. Currently ReinForest does not allow circles in the graph, so that the knowledge ontology forms a Direct Acyclic Graph (DAG). In practise, the dependencies encode the domain knowledge. For example, in order to know about the weather, the agent needs to ask date and location first.

**Attribute:** An attribute is the basic memory unit of a concept. It is an object class that contains:

1. Key: the unique ID of the attribute.
2. Value: the raw value from user input.
3. Normalized Value: the value that is normalized (e.g. today → 4/4/2016)
4. Score: a score indicates the confidence level.

**Concept Pool:** Given the definition of a concept, ReinForest allows developers to construct domain knowledge in the form of a DAG. ReinForest also introduces the idea of concept pool to create groups of concepts. Figure 4 gives a simple example of a concept pool for a slot-filling dialog manager. Essentially a concept pool is a collection of concepts that share some common properties. In the default setting, there are two concept pools: the agent concept pool and the user concept pool. The agent concept pool consists of concepts that the agent knows, such as agent’s name, the restaurant information and etc. On the other hand, the user concept pool contains concepts that only users know, such as the date that the users are inquiring about, users’ names etc.

### 6 Dialog Engine

As illustrated in Figure 2, the dialog engine of ReinForest is a domain-independent execution mechanism that consists of 4 main components: hierarchical policy execution, belief update, tree transformation and error handling. Algorithm 1 sketches the pseudo code of the main execution loop inside ReinForest. The dialog engine establishes its connection to the knowledge ontology via a dialog state, \( s \), which captures all the information about the ongoing dialogs. The agent/user concept pools are both parts of the dialog state \( s \). Now we formally define the dialog state and each of these four modules in the dialog engine.
Figure 4: A simple slot-filling dialog manager. The agent knows about its own name and restaurant. In order to inform about restaurant information, it needs to acquire three concepts: price, location and type from the users. Therefore, the restaurant concept depends on the three concepts from the user concept pool.

Algorithm 1 ReinForest Main Loop

```
while dialog.end() ≠ True do
    while dialog_stack.top().type ≠ PRIMITIVE do
        execute(dialog_stack.top.agent)
    end while
    if user has input then
        belief_update()
        tree_transformation()
        error_handle()
    end if
end while
```

6.1 Dialog State

The dialog state contains a pointer to the knowledge ontology as well as extra state information about a dialog. The current implementation of ReinForest contains simple variables that are useful to make decisions. Some example state variables include: number of turns, previous utterance annotation, previous selected domain etc.

6.2 Hierarchical Policy

Hierarchical policy has been studied extensively in the literature of both hierarchical reinforcement learning (HRL) [5, 8, 10] and plan-based dialog management [2, 4]. The contribution of ReinForest is that it formalizes the plan-based dialog management in the language of HRL, which opens up the possibility of applying well-established HRL algorithms to optimize the operations of plan-based dialog manager. We first introduce the notations of HRL and then define the dialog task tree using that language.

**Hierarchical Reinforcement Learning:** The mathematical framework of HRL is the Markov Decision Process (MDP) which is described by:

1. $S$ is the dialog state space
2. $A$ is a set of primitive actions.
3. $P(s'|s,a)$ defines the transition probability of executing primitive action $a$ in state, $s$
4. $R(s'|s,a)$ is the reward function defined over $S$ and $A$
An MDP, $M$ can be decomposed into a set of subtasks, $O = \{O_1, O_2, \ldots, O_k\}$, where $O_0$, by convention, is the root and solving $O_0$ solves $M$. A subtask is a Semi-Markov Decision Process (SMDP), which is characterised by two additional variables compared to an MDP:

1. $\beta_i(s)$ is the termination predicate of subtask $O_i$ that partitions $S$ into a set of active states, $S_i$ and a set of terminal states $T_i$. If $O_i$ enters a state in $T_i$, $O_i$ and its descendants exit immediately, i.e. $\beta_i(s) = 1$ if $s \in T_i$, otherwise $\beta_i(s) = 0$.

2. $U_i$ is a nonempty set of actions that can be performed by $O_i$. The actions can be either primitive actions from $A$ or other subtasks, $O_j$, where $i \neq j$. We will refer to $U_i$ as the children of subtask $O_i$.

Eventually, a hierarchical policy for $M$ is $\pi$ and is simply a set of policies for each subtask, i.e. $\pi = \{\pi_0, \pi_1, \ldots, \pi_n\}$. It is evident that a valid hierarchical policy forms a DAG, whose roots are from $O_0$ and where terminal leaves are primitive actions belonging to $A$.

**ReinForest Dialog Policy:** ReinForest Dialog Policy forms a dialog task tree. Each node is a subtask belonging to one the three types: dialog agency, dialog choice agency and dialog agent.

- Dialog Agency: a subtask $O_i \in O$ with a fixed policy that will execute its children from left to right.
- Dialog Choice Agency: a subtask $O_i \in O$ with a learned policy that chooses the next executed child based on the context.
- Dialog Agent: a primitive action $a \in A$ that actually delivers the action to users.

For visual illustration, Figure 5 shows our notation of expressing dialog task tree made from the above 3 basic types of nodes.

![Figure 5: Visual notation for the 3 basic types of nodes in the ReinForest Dialog Task Tree.](image)

6.3 **Belief Update**

Belief update takes place when there is new input from the users. This component will first update generic dialog states, such as incrementing the dialog turn count. Then it loops through all the concepts in the knowledge ontology and checks if the annotations in the new input match with any subscribed domain/intent/entities in each concept. If a match is found, the new values are stored in the attribute map of the concept.

6.4 **Tree Transformation**

Tree transformation is key in ReinForest in order to support mixed-initiative and multi-domain conversations. The transformation is comprised of two steps: candidate tree generation and candidate tree selection.

**Candidate tree generation:** scans through the updates made by belief update and generates a list of candidate trees that can be pushed to the dialog stack (and can be $\emptyset$). Usually a candidate is generated when the user explicitly requests information that is handled by a different domain.

**Candidate selection:** the selected candidate trees are appended under a dialog choice agency. The dialog choice agency selects one of the trees and pushes it to the dialog stack.
6.5 Error Handling

There are two types of error handling: **misunderstand error handling** and **non-understand error handling** [2]. Misunderstand handling is used to conduct explicit or implicit confirms about concepts in the knowledge ontology. Specifically, the dialog engine will loop through all concepts in the user concept pool and select concepts that are updated but not yet grounded for misunderstanding error handling. A misunderstanding subtask will then be pushed to the stack. The misunderstanding subtask will choose a built-in misunderstanding error handling strategy to confirm each concept. Current implementation supports two types of strategies: implicit and explicit confirm.

On the other hand, non-understanding handling is activated when there is user input, but no update or tree-transformation is able to succeed. ReinForest implements a wide range of non-understand handling strategies, ranging from simple "can you repeat that?" to a response from an external chat-bot.

7 Useful Policies

During development, several reusable and domain-independent subtasks are identified. This section briefly describes their use cases.

**Inform Tree:** Inform tree is shown in Figure 6 and is used for informing a concept from the agent concept pool. The inform root is a dialog agency so that it executes its children in order from left to right. The left branch is a sub-tree that will be recursively constructed to acquire all the dependent concepts.

**Request Tree:** Request tree is shown in Figure 6. It is a simple tree rooted with a dialog choice agency. Its children contain every dimension of a concept. Since the root is a choice agency, the execution order depends on the specific policy.

**Misunderstanding Tree:** shown in Figure 7 is a simple implementation of misunderstanding error handling. The root has a list of dialog choice agencies, each of them is in charge of confirming a specific concept. Each choice agency will choose a misunderstanding strategy for each concept.

**Non-understanding Tree:** shown in Figure 7 is implemented as a dialog choice agency with a list of non-understanding strategies as children. This tree is invoked when the new user input cannot be understood completely. Currently ReinForest implements 4 different non-understanding strategies: *Ask Repeat*, *Ask Rephrase*, *Notify Non-understand* and *a Chat Bot*.

8 Conclusion

In conclusion, this report demonstrates a simple framework, ReinForest, for rapid multi-domain dialog manager development. We introduce the interface and the architecture of ReinForest and formally describe the two core components: knowledge ontology and dialog engine. At last, we share useful and reusable dialog policies that we found during development.
Future work include using the framework to develop sophisticated multi-domain dialog systems and automate each components using machine learning methods.

References


