Feudal Reinforcement Learning for Dialogue Management in Large Domains

Iñigo Casanueva[1*], **Paweł Budzianowski**[1], **Pei-Hao Su**[2],
Stefan Ultes[1], **Lina Rojas-Barahona**[1], **Bo-Hsiang Tseng**[1] **and Milica Gašić**[1]
[1]Department of Engineering, University of Cambridge, UK
[2]PolyAI Limited, London, UK
ic340@cam.ac.uk

Abstract

Reinforcement learning (RL) is a promising approach to solve dialogue policy optimisation. Traditional RL algorithms, however, fail to scale to large domains due to the curse of dimensionality. We propose a novel Dialogue Management architecture, based on Feudal RL, which decomposes the decision into two steps; a first step where a master policy selects a subset of primitive actions, and a second step where a primitive action is chosen from the selected subset. The structural information included in the domain ontology is used to abstract the dialogue state space, taking the decisions at each step using different parts of the abstracted state. This, combined with an information sharing mechanism between slots, increases the scalability to large domains. We show that an implementation of this approach, based on Deep-Q Networks, significantly outperforms previous state of the art in several dialogue domains and environments, without the need of any additional reward signal.

1 Introduction

Task-oriented Spoken Dialogue Systems (SDS), in the form of personal assistants, have recently gained much attention in both academia and industry. One of the most important modules of a SDS is the Dialogue Manager (DM) (or policy), the module in charge of deciding the next action in each dialogue turn. Reinforcement Learning (RL) (Sutton and Barto, 1999) has been studied for several years as a promising approach to model dialogue management (Levin et al., 1998; Henderson et al., 2008; Pietquin et al., 2011; Young et al., 2013; Casanueva et al., 2015; Su et al., 2016). However, as the dialogue state space increases, the number of possible trajectories needed to be explored grows exponentially, making traditional RL methods not scalable to large domains.

Hierarchical RL (HRL), in the form of temporal abstraction, has been proposed in order to mitigate this problem (Cuayáhuitl et al., 2010, 2016; Budzianowski et al., 2017; Peng et al., 2017). However, proposed HRL methods require that the task is defined in a hierarchical structure, which is usually handcrafted. In addition, they usually require additional rewards for each subtask. Space abstraction, instead, has been successfully applied to dialogue tasks such as Dialogue State Tracking (DST) (Henderson et al., 2014b), and policy transfer between domains (Gašić et al., 2013, 2015; Wang et al., 2015). For DST, a set of binary classifiers can be defined for each slot, with shared parameters, learning a general way to track slots. The policy transfer method presented in (Wang et al., 2015), named Domain Independent Parametrisation (DIP), transforms the belief state into a slot-dependent fixed size representation using a handcrafted feature function. This idea could also be applied to large domains, since it can be used to learn a general way to act in any slot.

In slot-filling dialogues, a HRL method that relies on space abstraction, such as Feudal RL (FRL) (Dayan and Hinton, 1993), should allow RL scale to domains with a large number of slots. FRL divides a task spatially rather than temporally, decomposing the decisions in several steps and using different abstraction levels in each sub-decision. This framework is especially useful in RL tasks with large discrete action spaces, making it very attractive for large domain dialogue management.

In this paper, we introduce a Feudal Dialogue Policy which decomposes the decision in each turn into two steps. In a first step, the policy decides if it takes a slot independent or slot dependent action. Then, the state of each slot sub-policy is abstracted to account for features related to that slot,

[*] Currently at PolyAI, inigo@poly-ai.com

Proceedings of NAACL-HLT 2018, pages 714–719
New Orleans, Louisiana, June 1 - 6, 2018. ©2018 Association for Computational Linguistics

and a primitive action is chosen from the previously selected subset. Our model does not require any modification of the reward function and the hierarchical architecture is fully specified by the structured database representation of the system (i.e. the ontology), requiring no additional design.

2 Background

Dialogue management can be cast as a continuous MDP (Young et al., 2013) composed of a continuous multivariate belief state space $\mathcal{B}$, a finite set of actions $\mathcal{A}$ and a reward function $R(b_t, a_t)$. At a given time t, the agent observes the belief state $b_t \in \mathcal{B}$, executes an action $a_t \in \mathcal{A}$ and receives a reward $r_t \in \mathbb{R}$ drawn from $R(b_t, a_t)$. The action taken, a, is decided by the *policy*, defined as the function $\pi(b) = a$. For any policy π and $b \in \mathcal{B}$, the Q-value function can be defined as the expected (discounted) return R, starting from state b, taking action a, and then following policy π until the end of the dialogue at time step T:

$$Q^{\pi}(b, a) = \mathbb{E}\{R|b_t = b, a_t = a\} \qquad (1)$$

where $R = \sum_{\tau=t}^{T-1} \gamma^{(\tau-t)} r_\tau$ and γ is a discount factor, with $0 \leq \gamma \leq 1$.

The objective of RL is to find an optimal policy π^*, i.e. a policy that maximizes the expected return in each belief state. In *Value-based* algorithms, the optimal policy can be found by greedily taking the action which maximises $Q^{\pi}(b, a)$.

In slot-filling SDSs the belief state space $\mathcal{B}$ is defined by the *ontology*, a structured representation of a database of entities that the user can retrieve by talking to the system. Each entity has a set of properties, refereed to as *slots* $\mathcal{S}$, where each of the slots can take a value from the set $\mathcal{V}_s$. The belief state b is then defined as the concatenation of the probability distribution of each slot, plus a set of general features (e.g. the communication function used by the user, the database search method...) (Henderson et al., 2014a). The set $\mathcal{A}$ is defined as a set of *summary actions*, where the actions can be either slot dependent (e.g. *request*(food), *confirm*(area)...) or slot independent[1] (e.g. *hello*(), *inform*()...).

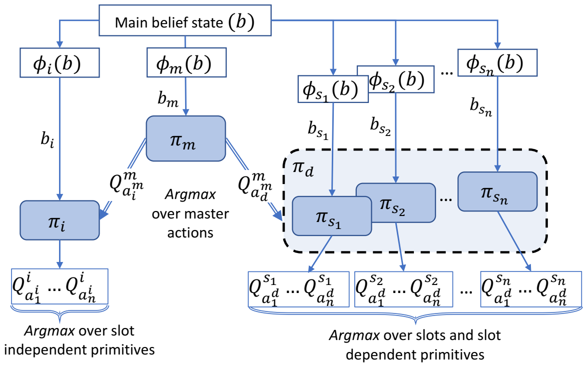
Main belief state (b)
$\phi_i(b)$
$\phi_m(b)$
$\phi_{s_1}(b)$
$\phi_{s_2}(b)$
...
$\phi_{s_n}(b)$
b_m
b_i
b_{s_1}
b_{s_2}
b_{s_n}
π_m
$Q^m_{a^m_i}$
$Q^m_{a^m_d}$
Argmax over master actions
π_i
π_d
π_{s_1}
π_{s_2}
...
π_{s_n}
$Q^i_{a^i_1}$ ... $Q^i_{a^i_n}$
$Q^{s_1}_{a^d_1}$ ... $Q^{s_1}_{a^d_n}$
$Q^{s_2}_{a^d_1}$ ... $Q^{s_2}_{a^d_n}$
...
$Q^{s_n}_{a^d_1}$ ... $Q^{s_n}_{a^d_n}$
Argmax over slot independent primitives
Argmax over slots and slot dependent primitives

Figure 1: Feudal dialogue architecture used in this work. The sub-policies surrounded by the dashed line have shared parameters. The simple lines show the data flow and the double lines the sub-policy decisions.

The belief space $\mathcal{B}$ is defined by the ontology, therefore belief states of different domains will have different shapes. In order to transfer knowledge between domains, Domain Independent Parametrization (DIP) (Wang et al., 2015) proposes to abstract the belief state b into a fixed size representation. As each action is either slot independent or dependent on a slot s, a feature function $\phi_{dip}(b, s)$ can be defined, where $s \in \mathcal{S} \cup s_i$ and s_i stands for slot independent actions. Therefore, in order to compute the policy, $Q(b, a)$ can be approximated as $Q(\phi_{dip}(b, s), a)$, where s is the slot associated to action a.

Wang et al. (2015) presents a handcrafted feature function $\phi_{dip}(b, s)$. It includes the slot independent features of the belief state, a summarised representation of the joint belief state, and a summarised representation of the belief state of the slot s. Section 4 gives a more detailed description of the $\phi_{dip}(b, s)$ function used in this work.

3 Feudal dialogue management

FRL decomposes the policy decision $\pi(b) = a$ in each turn into several sub-decisions, using different abstracted parts of the belief state in each sub-decision. The objective of a task oriented SDS is to fulfill the users goal, but as the goal is not observable for the SDS, the SDS needs to gather enough information to correctly fulfill it. Therefore, in each turn, the DM can decompose its decision in two steps: first, decide between taking an action in order to gather information about the user goal (information gathering actions) or taking an action to fulfill the user goal or a part of it (information providing actions) and second, select a (primitive) action to execute from the previously selected subset. In a slot-filling dialogue, the set of

[1] We include the summary actions dependent on all the slots, such as *inform*(), in this group.

information gathering actions can be defined as the set of slot dependent actions, while the set of information providing actions can be defined as the remaining actions.

The architecture of the feudal policy proposed by this work is represented schematically in Figure 1. The (primitive) actions are divided between two subsets; slot independent actions $\mathcal{A}_i$ (e.g. hello(), inform()); and slot dependent actions $\mathcal{A}_d$ (e.g. request(), confirm())[2]. In addition, a set of master actions $\mathcal{A}_m = (a^m_i, a^m_d)$ is defined, where a^m_i corresponds to taking an action from $\mathcal{A}_i$ and a^m_d to taking an action from $\mathcal{A}_d$. Then, a feature function $\phi_s(b) = b_s$ is defined for each slot $s \in \mathcal{S}$, as well as a slot independent feature function $\phi_i(b) = b_i$ and a master feature function $\phi_m(b) = b_m$. These feature functions can be handcrafted (e.g. the DIP feature function introduced in section 2) or any function approximator can be used (e.g. neural networks trained jointly with the policy).

Finally, a master policy $\pi_m(b_m) = a^m$, a slot independent policy $\pi_i(b_i) = a^i$ and a set of slot specific policies $\pi_s(b_s) = a^d$, one for each $s \in \mathcal{S}$, are defined, where $a^m \in \mathcal{A}_m$, $a^i \in \mathcal{A}_i$ and $a^d \in \mathcal{A}_d$. Contrary to other feudal policies, the slot specific sub-policies have shared parameters, in order to generalise between slots (following the idea used by Henderson et al. (2014b) for DST). The differences between the slots (size, value distribution...) are accounted by the feature function $\phi_s(b)$. Therefore $\pi_m(b_m)$ is defined as:

$$\pi_m(b_m) = \underset{a^m \in \mathcal{A}_m}{\operatorname{argmax}}\, Q^m(b_m, a^m) \tag{2}$$

If $\pi_m(b_m) = a^m_i$, the sub-policy run is π_i:

$$\pi_i(b_i) = \underset{a^i \in \mathcal{A}_i}{\operatorname{argmax}}\, Q^i(b_i, a^i) \tag{3}$$

Else, if $\pi_m(b_m) = a^m_d$, π_d is selected. This policy runs each slot specific policy, π_s, for all $s \in \mathcal{S}$, choosing the action-slot pair that maximises the Q function over all the slot sub-policies.

$$\pi_d(b_s | \forall s \in \mathcal{S}) = \underset{a^d \in \mathcal{A}_d, s \in \mathcal{S}}{\operatorname{argmax}}\ Q^s(b_s, a^d) \tag{4}$$

Then, the summary action a is constructed by joining a^d and s (e.g. if a^d=*request()* and s=*food*, then the summary action will be *request(food)*). A pseudo-code of the Feudal Dialogue Policy algorithm is given in Appendix A.

Domain	Code	# constraint slots	# requests	# values
Cambridge Restaurants	CR	3	9	268
San Francisco Restaurants	SFR	6	11	636
Laptops	LAP	11	21	257

	Env. 1	Env. 2	Env. 3	Env. 4	Env. 5	Env. 6
SER	0%	0%	15%	15%	15%	30%
Masks	on	off	on	off	on	on
User	Std.	Std.	Std.	Std.	Unf.	Std.

Table 1: Sumarised description of the domains and environments used in the experiments. Refer to (Casanueva et al., 2017) for a detailed description.

4 Experimental setup

The models used in the experiments have been implemented using the PyDial toolkit (Ultes et al., 2017)[3] and evaluated on the PyDial benchmarking environment (Casanueva et al., 2017). This environment presents a set of tasks which span different size domains, different Semantic Error Rates (SER), and different configurations of action masks and user model parameters (Standard (Std.) or Unfriendly (Unf.)). Table 1 shows a sumarised description of the tasks. The models developed in this paper are compared to the state-of-the-art RL algorithms and to the handcrafted policy presented in the benchmarks.

4.1 DIP-DQN baseline

An implementation of DIP based on Deep-Q Networks (DQN) (Mnih et al., 2013) is implemented as an additional baseline (Papangelis and Stylianou, 2017). This policy, named DIP-DQN, uses the same hyperparameters as the DQN implementation released in the PyDial benchmarks. A DIP feature function based in the description in (Wang et al., 2015) is used, $\phi_{dip}(b, s) = \psi_0(b) \oplus \psi_j(b) \oplus \psi_d(b, s)$, where:
• $\psi_0(b)$ accounts for general features of the belief state, such as the database search method.
• $\psi_j(b)$ accounts for features of the joint belief state, such as the entropy of the joint belief.
• $\psi_d(b, s)$ accounts for features of the marginal distribution of slot s, such as the entropy of s.
Appendix B shows a detailed description of the DIP features used in this work.

4.2 Feudal DQN policy

A Feudal policy based on the architecture described in sec. 3 is implemented, named FDQN. Each sub-policy is constructed by a DQN policy

[2] Note that the actions of this set are composed just by the communication function of the slot dependent actions, thus reducing the number of actions compared to $\mathcal{A}$.

[3] The implementation of the models can be obtained in www.pydial.org

		Feudal-DQN		DIP-DQN		Bnch.	Hdc.
	Task	Suc.	Rew.	Suc.	Rew.	Rew.	Rew.
Env. 1	CR	89.3%	11.7	48.8%	-2.8	13.5	**14.0**
	SFR	71.1%	7.1	25.8%	-7.4	11.7	**12.4**
	LAP	65.5%	5.7	26.6%	-8.8	10.5	**11.7**
Env. 2	CR	97.8%	13.1	85.5%	9.6	12.2	**14.0**
	SFR	95.4%	**12.4**	85.7%	8.4	9.6	**12.4**
	LAP	94.1%	**12.0**	89.5%	9.7	7.3	11.7
Env. 3	CR	92.6%	11.7	86.1%	8.9	**11.9**	11.0
	SFR	90.0%	**9.7**	59.3%	0.2	8.6	9.0
	LAP	89.6%	**9.4**	71.5%	3.1	6.7	8.7
Env. 4	CR	91.4%	**11.2**	82.6%	8.7	10.7	11.0
	SFR	90.3%	**10.2**	86.1%	9.2	7.7	9.0
	LAP	88.7%	**9.8**	74.8%	6.0	5.5	8.7
Env. 5	CR	96.3%	**11.5**	74.4%	2.9	10.5	9.3
	SFR	88.9%	**7.9**	75.5%	3.2	4.5	6.0
	LAP	78.8%	5.2	64.4%	-0.4	4.1	**5.3**
Env. 6	CR	90.6%	**10.4**	83.4%	8.1	10.0	9.7
	SFR	83.0%	**7.1**	71.9%	3.9	3.9	6.4
	LAP	78.5%	**6.0**	66.5%	2.7	3.6	5.5

Table 2: Success rate and reward for Feudal-DQN and DIP-DQN in the 18 benchmarking tasks, compared with the reward of the best performing algorithm in each task (Bnch.) and the handcrafted policy (Hdc.) presented in (Casanueva et al., 2017).

(Su et al., 2017). These policies have the same hyperparameters as the baseline DQN implementation, except for the two hidden layer sizes, which are reduced to 130 and 50 respectively. As feature functions, subsets of the DIP features are used:

$$\phi_m(b) = \phi_i(b) = \psi_0(b) \oplus \psi_j(b)$$
$$\phi_s(b) = \psi_0(b) \oplus \psi_j(b) \oplus \psi_d(b, s) \, \forall s \in \mathcal{S}$$

The original set of summary actions of the benchmarking environment, $\mathcal{A}$, has a size of $5 + 3 * |\mathcal{S}|$, where $|\mathcal{S}|$ is the number of slots. This set is divided in two subsets[4]: $\mathcal{A}_i$ of size 6 and $\mathcal{A}_d$ of size 4. Each sub-policy (including π_m) is trained with the same sparse reward signal used in the baselines, getting a reward of 20 if the dialogue is successful or 0 otherwise, minus the dialogue length.

5 Results

The results in the 18 tasks of the benchmarking environment after 4000 training dialogues are presented in Table 2. The same evaluation procedure of the benchmarks is used, presenting the mean over 10 different random seeds and testing every seed for 500 dialogues. The FDQN policy substantially outperforms every other other policy in all the environments except Env. 1. The

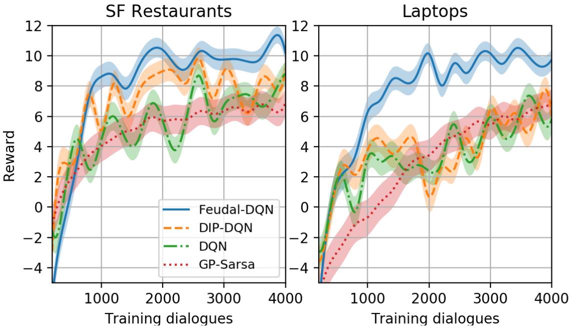

Figure 2: Learning curves for Feudal-DQN and DIP-DQN in Env. 4, compared to the two best performing algorithms in (Casanueva et al., 2017) (DQN and GP-Sarsa). The shaded area depicts the mean ± the standard deviation over ten random seeds.

performance increase is more considerable in the two largest domains (SFR and LAP), with gains up to 5 points in accumulated reward in the most challenging environments (e.g. Env. 4 LAP), compared to the best benchmarked RL policies (Bnch.). In addition, FDQN consistently outperforms the handcrafted policy (Hdc.) in environments 2 to 6, which traditional RL methods could not achieve. In Env. 1, however, the results for FDQN and DIP-DQN are rather low, specially for DIP-DQN. Surprisingly, the results in Env. 2, which only differs from Env. 1 in the absence of action masks (thus, in principle, is a more complex environment), outperform every other algorithm. Analysing the dialogues individually, we could observe that, in this environment, both policies are prone to "overfit" to an action [5]. The performance of FDQN and DIP-DQN in Env. 4 is also better than in Env. 3, while the difference between these environments also lies in the masks. This suggests that an specific action mask design can be helpful for some algorithms, but can harm the performance of others. This is especially severe in the DIP-DQN case, which shows good performance in some challenging environments, but it is more unstable and prone to overfit than FDQN.

However, the main purpose of action masks is to reduce the number of dialogues needed to train a policy. Observing the learning curves shown in Figure 2, the FDQN model can learn a near-optimal policy in large domains in about 1500 dialogues, even if no additional reward is used, making the action masks unnecessary.

[4] An additional *pass()* action is added to each subset, which is taken whenever the other sub-policy is executed. This simplifies the training algorithm.

[5] The model overestimates the value of an incorrect action, continuously repeating it until the user runs out of patience.

6 Conclusions and future work

We have presented a novel dialogue management architecture, based on Feudal RL, which substantially outperforms the previous state of the art in several dialogue environments. By defining a set of slot dependent policies with shared parameters, the model is able to learn a general way to act in slots, increasing its scalability to large domains.

Unlike other HRL methods applied to dialogue, no additional reward signals are needed and the hierarchical structure can be derived from a flat ontology, substantially reducing the design effort.

A promising approach would be to substitute the handcrafted feature functions used in this work by neural feature extractors trained jointly with the policy. This would avoid the need to design the feature functions and could be potentially extended to other modules of the SDS, making text-to-action learning tractable. In addition, a single model can be potentially used in different domains (Papangelis and Stylianou, 2017), and different feudal architectures could make larger action spaces tractable (e.g. adding a third sub-policy to deal with actions dependent on 2 slots).

Acknowledgments

This research was funded by the EPSRC grant EP/M018946/1 Open Domain Statistical Spoken Dialogue Systems.

References

Paweł Budzianowski, Stefan Ultes, Pei-Hao Su, Nikola Mrkšić, Tsung-Hsien Wen, Inigo Casanueva, Lina M. Rojas Barahona, and Milica Gašić. 2017. Subdomain modelling for dialogue management with hierarchical reinforcement learning. In *Proc of SIGDIAL*.

Iñigo Casanueva, Paweł Budzianowski, Pei-Hao Su, Nikola Mrkšić, Tsung-Hsien Wen, Stefan Ultes, Lina Rojas-Barahona, Steve Young, and Milica Gašić. 2017. A benchmarking environment for reinforcement learning based task oriented dialogue management. *arXiv preprint arXiv:1711.11023* .

Inigo Casanueva, Thomas Hain, Heidi Christensen, Ricard Marxer, and Phil Green. 2015. Knowledge transfer between speakers for personalised dialogue management. In *Proceedings of the 16th Annual Meeting of the Special Interest Group on Discourse and Dialogue*. pages 12–21.

Heriberto Cuayáhuitl, Steve Renals, Oliver Lemon, and Hiroshi Shimodaira. 2010. Evaluation of a hierarchical reinforcement learning spoken dialogue system. *Computer Speech & Language* 24(2):395–429.

Heriberto Cuayáhuitl, Seunghak Yu, Ashley Williamson, and Jacob Carse. 2016. Deep reinforcement learning for multi-domain dialogue systems. *NIPS Workkshop* .

Peter Dayan and Geoffrey E Hinton. 1993. Feudal reinforcement learning. In *Advances in neural information processing systems*. pages 271–278.

Milica Gašić, Catherine Breslin, Matthew Henderson, Dongho Kim, Martin Szummer, Blaise Thomson, Pirros Tsiakoulis, and Steve Young. 2013. Pomdp-based dialogue manager adaptation to extended domains. In *Proceedings of the SIGDIAL Conference*.

Milica Gašić, Nikola Mrkšić, Pei-hao Su, David Vandyke, Tsung-Hsien Wen, and Steve Young. 2015. Policy committee for adaptation in multi-domain spoken dialogue systems. In *Automatic Speech Recognition and Understanding (ASRU), 2015 IEEE Workshop on*. IEEE, pages 806–812.

James Henderson, Oliver Lemon, and Kallirroi Georgila. 2008. Hybrid reinforcement/supervised learning of dialogue policies from fixed data sets. *Computational Linguistics* 34(4):487–511.

M. Henderson, B. Thomson, and J. Williams. 2014a. The Second Dialog State Tracking Challenge. In *Proc of SIGdial*.

M. Henderson, B. Thomson, and S. J. Young. 2014b. Word-based Dialog State Tracking with Recurrent Neural Networks. In *Proc of SIGdial*.

Esther Levin, Roberto Pieraccini, and Wieland Eckert. 1998. Using markov decision process for learning dialogue strategies. In *Acoustics, Speech and Signal Processing, 1998. Proceedings of the 1998 IEEE International Conference on*. IEEE, volume 1, pages 201–204.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* .

Alexandros Papangelis and Yannis Stylianou. 2017. Single-model multi-domain dialogue management with deep learning. In *International Workshop for Spoken Dialogue Systems*.

B. Peng, X. Li, L. Li, J. Gao, A. Celikyilmaz, S. Lee, and K.-F. Wong. 2017. Composite Task-Completion Dialogue System via Hierarchical Deep Reinforcement Learning. *ArXiv e-prints* .

Olivier Pietquin, Matthieu Geist, Senthilkumar Chandramohan, and Hervé Frezza-Buet. 2011. Sample-efficient batch reinforcement learning for dialogue management optimization. *ACM Transactions on Speech and Language Processing (TSLP)* 7(3):7.

Pei-Hao Su, Pawel Budzianowski, Stefan Ultes, Milica Gasic, and Steve Young. 2017. Sample-efficient actor-critic reinforcement learning with supervised data for dialogue management. *Proceedings of SigDial* .

Pei-Hao Su, Milica Gasic, Nikola Mrksic, Lina Rojas-Barahona, Stefan Ultes, David Vandyke, Tsung-Hsien Wen, and Steve Young. 2016. Continuously learning neural dialogue management. *arXiv preprint arXiv:1606.02689* .

Richard S. Sutton and Andrew G. Barto. 1999. *Reinforcement Learning: An Introduction*. MIT Press.

Stefan Ultes, Lina M. Rojas-Barahona, Pei-Hao Su, David Vandyke, Dongho Kim, Iñigo Casanueva, Paweł Budzianowski, Nikola Mrkšić, Tsung-Hsien Wen, Milica Gašić, and Steve J. Young. 2017. Pydial: A multi-domain statistical dialogue system toolkit. In *ACL Demo*. Association of Computational Linguistics.

Zhuoran Wang, Tsung-Hsien Wen, Pei-Hao Su, and Yannis Stylianou. 2015. Learning domain-independent dialogue policies via ontology parameterisation. In *SIGDIAL Conference*. pages 412–416.

Steve Young, Milica Gašić, Blaise Thomson, and Jason D Williams. 2013. Pomdp-based statistical spoken dialog systems: A review. *Proceedings of the IEEE* 101(5):1160–1179.

A Feudal Dialogue Policy algorithm

Algorithm 1 Feudal Dialogue Policy

```
for each dialogue turn do
    observe $b$
    $b_m = \phi_m(b)$
    $a^m = \underset{a^m \in \mathcal{A}_m}{\text{argmax}}\, Q^m(b_m, a^m)$
    if $a^m == a^m_i$ then                          ▷ drop to $\pi_i$
        $b_i = \phi_i(b)$
        $a = \underset{a^i \in \mathcal{A}_i}{\text{argmax}}\, Q^i(b_i, a^i)$
    else $a^m == a^m_d$ then                        ▷ drop to $\pi_d$
        $b_s = \phi_s(b)\, \forall s \in \mathcal{S}$
        $slot, act = \underset{s \in \mathcal{S}, a^d \in \mathcal{A}_d}{\text{argmax}}\, Q^s(b_s, a^d)$
        $a = join(slot, act)$
    end if
    execute $a$
end for
```

B DIP features

This section gives a detailed description of the DIP feature functions $\phi_{dip}(b, s) = \psi_0(b) \oplus \psi_j(b) \oplus \psi_d(b, s)$ used in this work. The differences with the features used in (Wang et al., 2015) and (Papangelis and Stylianou, 2017) are the following:

- No *priority* or *importance* features are used.
- No *Potential contribution to DB search* features are used.
- The joint belief features $\psi_j(b)$ are extended to account for large-domain aspects.

Feature function	Feature description	Feature size
$\psi_0(b)$	last user dialogue act (bin) *	7
	DB search method (bin) *	6
	# of requested slots (bin)	5
	offer happened *	1
	last action was *Inform no venue* *	1
	normalised # of slots (1/# of slots)	1
	normalised avg. slot length (1/avg. # of values)	1
$\psi_j(b)$	prob. of the top 3 values of b_j	3
	prob. of *NONE* value of b_j	1
	entropy of b_j	1
	diff. between top and 2nd value probs. (bin)	5
	# of slots with top value not *NONE* (bin)	5
$\psi_d(b, s)$	prob. of the top 3 values of s	3
	prob. of *NONE* value of s	1
	diff. between top and 2nd value probs. (bin)	5
	entropy of s	1
	# of values of s with prob. > 0 (bin)	5
	normalised slot length (1/# of values)	1
	slot length (bin)	10
	entropy of the distr. of values of s in the DB	1
	total	64

Table 3: List of features composing the DIP features. the tag (bin) denotes that a binary encoding is used for this feature. Some of the joint features $\psi_j(b)$ are extracted from the joint belief b_j, computed as the Cartesian product of the beliefs of the individual slots. * denotes that these features exist in the original belief state b.