Towards a Virtual Personal Assistant Based on a User-Defined Portfolio of Multi-Domain Vocal Applications

Conference Paper · December 2016

DOI: 10.1109/SLT.2016.7846252

CITATIONS
0

READS
61

4 authors, including:

Tatiana Ekeinhor-Komi
Université d´Avignon et des Pays du Vaucluse
1 PUBLICATION **0** CITATIONS
SEE PROFILE

Jean-Léon Bouraoui
Orange Labs
44 PUBLICATIONS **89** CITATIONS
SEE PROFILE

Romain Laroche
Microsoft Maluuba
58 PUBLICATIONS **185** CITATIONS
SEE PROFILE

TOWARDS A VIRTUAL PERSONAL ASSISTANT BASED ON A USER-DEFINED PORTFOLIO OF MULTI-DOMAIN VOCAL APPLICATIONS

Tatiana Ekeinhor-Komi[1,2]*, Jean-Léon Bouraoui*[1]*, Romain Laroche*[1]*, Fabrice Lefèvre*[2]

[1] Orange Labs
[2] CERI-LIA, Université d'Avignon

firstname.surname@orange.com,firstname.surname@univ-avignon.fr

ABSTRACT

This paper proposes a novel approach to defining and simulating a new generation of virtual personal assistants as multi-application multi-domain distributed dialogue systems. The first contribution is the assistant architecture, composed of independent third-party applications handled by a Dispatcher. In this view, applications are black-boxes responding with a self-scored answer to user requests. Next, the Dispatcher distributes the current request to the most relevant application, based on these scores and the context (history of interaction etc.), and conveys its answer to the user. To address variations in the user-defined portfolio of applications, the second contribution, a stochastic model automates the online optimisation of the Dispatcher's behaviour. To evaluate the learnability of the Dispatcher's policy, several parametrisations of the user and application simulators are enabled, in such a way that they cover variations of realistic situations. Results confirm in all considered configurations of interest, that reinforcement learning can learn adapted strategies.

Index Terms— multi-application spoken dialogue systems, multi-domain, dialogue strategy, reinforcement learning.

1. INTRODUCTION

Virtual personal assistants (VPA) are dialogue systems allowing a large range of tasks. Such systems are said to be multi-domain [1, 2, 3]. In such systems, independent dialogue applications are cooperating in various ways. Most of the previous studies share to consider that domains and applications are strictly aligned. For instance, with two domains "weather" and "flight", there will only be one application (or module) related to each domain. However, this view is limited as there are situations in which applications, even acting on different domains, may share the same semantic concepts leading to ambiguity, semantic overlap and homophones. For example, a weather forecast application shares the city names with a flight reservation application. Conversely, one can imagine VPA in which several applications co-exist for the same domain (coming from different manufacturers for instance).

The ultimate goal would be to have each user define the set of dialogue applications she wants to install on her VPA, but in the current situation, it requires consistency in the application portfolio. Such constraint limits VPA to be almost exclusively designed by companies which are able to gather a large and extensible set of tightly integrated applications (either in-house or from third-party partners). In our proposition, potentially any application could be automatically added to the portfolio, after a light interface convention is accepted (mostly providing a self-scored answer). Then, an online trainable policy will allow the development of an assistant with a user-defined and extensible portfolio of applications.

The *distributed* architecture introduced in [4] is now commonly adopted in multi-domain dialogue systems [5, 2]. The primary challenge of such an architecture is thus to select, at the right time, the right application related to the right domain. Different methodologies already exist to tackle this problem. Amongst them, Supervised Learning methods are the most popular [4, 6, 1]. The selection is generally performed as a two-fold classification to decide whether to remain on the current application or switch to the one with the best score, if not the same. A three-fold classification, with the third category being 'out-of-domain', has also been considered [5]. Additionally, dialogue history has been used to improve the selection [7, 8, 9].

Yet, a dialogue is a complex task that also requires a clarification and error-handling strategy. Indeed, clarification plays a major role in multi-domain dialogue due to domain overlaps, disambiguation of the user's intent or wrong assumptions about domains. Reinforcement Learning (RL) [10] based systems [11, 12, 13] are also considered where dialogue is seen as an optimization problem leading to optimal strategies. Unlike local classification, RL offers the advantage of considering the dialogue as a sequence of action selection to achieve a long term goal and can therefore integrate clarification steps for the sake of the whole dialogue success.

One shortcoming of these methods though, is that they require a huge amount of data of user interactions. As a workaround, many studies have proven that a system can be bootstrapped by using a user simulator [14, 15, 16], which can

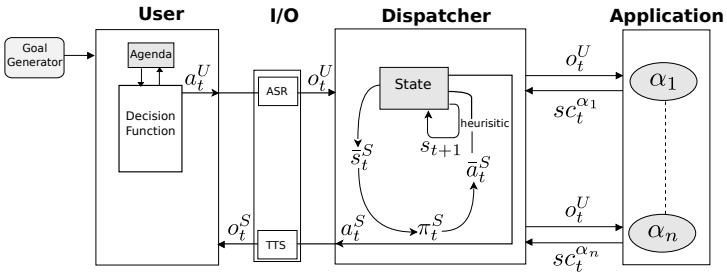

Fig. 1. *VPA simulator architecture*

act	arg	Description
offer	α_i	transfers the answer of the application α_i
confExpl	α_i	explicitly asks for confirmation about the choice α_i
askSelect	$\alpha_i, \alpha_{i'}$	asks the user to choose between two options α_i and $\alpha_{i'}$
askRepeat	$\emptyset$	asks the user to repeat
bye	$\emptyset$	ends the dialogue

Table 1. Dispatcher dialogue acts

also be used for design automation and optimization [17, 18]. Lacking available representative data of the proposed scheme, we have recourse to simulation as a proof-of-concept in this paper. It gives us the opportunity to develop a first implementation of the Dispatcher architecture and we define the configuration variables in the simulation setup to be able to render some realistic situations. Based on these situations, RL training is shown to outperform handcrafted versions of the Dispatcher strategy for handling applications better able to answer users' requests.

Section 2 presents our first contribution: the design of a novel distributed architecture enabling the automatic fusion of black box dialogue applications into a multi-domain system. These dialogue applications are only required to return a score for their own relevance to answer the current request. Next, Sections 3 and 4 expose our second contribution: the abstraction model for distributions of applications over domains, inter-domain confusions, and understanding error model allowing the development of a simulator of the proposed architecture. In Section 5, we report our third contribution: the simulation results confirming that a Dispatcher's policy can be RL-optimized to highly variable configuration setups. Finally, Section 6 concludes the article.

2. INTERACTION MODEL

A human-machine dialogue is a sequence of interactions between a user and a system. Figure 1 depicts the proposed VPA architecture. Let n be the number of applications in the current VPA configuration and m be the number of underlying domains. Then, the system's service is embodied in a portfolio of applications $\{\alpha_i\}_{i\in[\![1,n]\!]}$, which handle domains $\{\delta_j\}_{j\in[\![1,m]\!]}$.

The user performs an action a_t^U in agreement with her current goal seen as a sequence of elementary tasks to perform. At every turn, the Dispatcher receives an observation o_t^U and sends it to all applications. Every application outputs its responses with a relevance score. Then, the Dispatcher performs an action a_t^S which consist in giving the ground to an application α^*, returning its answer to the user or in entering a clarification stage. The user, in her turn, observes o_t^S.

The exchanges are formalised through dialogue acts (DAs). Each DA is defined as $\texttt{act}(arg)$ where act denotes the DA type and arg denotes its argument list, as inspired by [19]. In the Dispatcher, DAs are only represented at the level of the targeted applications or domains. Table 1 reports the list of system actions (user DAs in the simulator are also abstracted, see Section 3.2).

3. SIMULATOR FRAMEWORK

Here, the different components of the simulator framework and their functions in our VPA architecture are detailed.

3.1. Goal Generator

Simulated user goals are generated by a goal generator. A goal is defined as a sequence of tasks: $G = \{\tau_k\}$. For instance, in order to prepare a trip, one has to check the flights, then the hotels, the places to visit, the weather forecast. The goal generation is characterized by two parameters: p_G influences the number of tasks in the goal and p_τ influences the type of each task.

The goal's length is sampled from a geometric distribution with parameter p_G. A task can have two types: tasks that should be solved directly by a specific application and tasks that can be resolved by any application handling a specific domain. In practice, a user may want to check any available flights for a defined travel and a particular schedule (flight domain) or, she may want to check exclusively the availability of Air France flights (Air France application). In the randomly generated goals, the type of a task is sampled according to the value of p_τ: the task is about an application with probability p_τ or a domain with probability $1 - p_\tau$. In this first setup, it is assumed that the tasks the user wants to perform are only about the applications or the domains available in the current configuration of the system. The management of out-of-domain requests is left for future work.

3.2. User simulator

The user behaviour is simulated at two levels: her intention in using the personal assistant, and her linguistic production.

At the beginning of each dialogue, a generated goal is attributed to the user. This goal is transformed into an agenda: each task τ_k requires one or several dialogue turns to be completed, and therefore requires many "ask" DAs to be performed. As for the goal length, each task length is sampled from a geometric distribution of parameter p_{Ag}. As a consequence, the agenda's length has an expected value of $1/p_G p_{Ag}$.

This agenda-based user simulation is inspired by [16]. But unlike [16] which defines the agenda in terms of constraints and requests, we remain at the intention level to perform tasks on applications or domains. Consequently, the agenda contains all the ask DAs the user wants to convey to the system. Table 2 contains the list of possible user DAs. Also for the sake of simplicity, we assume that during the dialogue, the user's goal does not change, though answers to some intermediate tasks can influence further ones (but in the real setting user's agenda is a latent variable anyway).

act	arg	**Description**
ask	$\alpha_i \vert \delta_j$	the user's request is expected to be handled by application α_i or by any application of domain δ_j
select	α_i	the user's request specifies explicitly the targeted application α_i
agree	α_i	the user agrees with the system's proposal α_i
refuse	α_i	the user does not agree with the system's proposal α_i
hangup	$\emptyset$	the user ends the dialogue before reaching her goal
bye	$\emptyset$	the user ends the dialogue

Table 2. Simulated user dialogue acts

Once the agenda has been defined, the user model is defined as a rule-based decision function selecting the user's actions. It is responsible for maintaining the negotiation exchange between the user and the system until the dialogue ends or fails. The dialogue fails when the user reaches the limit of her patience. The patience is defined as the capacity of the user to tolerate the system errors and misunderstandings. We distinguish the patience at the task level (*local patience*) and at the dialogue level (*global patience*). The former indicates the maximal number of turns necessary for the user to move to another task even if the current one is not achieved. The latter is the total number of errors the user tolerates before she ends the dialogue irrespective of reaching her goal.

An action from the user agenda a_t^U may either concern an application α_i or a domain δ_j. In this latter case any application that belongs to this domain is a relevant decision. The action a_t^U is represented as an m-size vector. Each domain manipulates a set of semantic concepts and applications are grouped by concept similarity. For a given task, the user will use specific semantic concepts. Consequently, we choose to represent a user utterance by an m-size vector, each dimension being a domain.

3.3. I/O error simulator

Input/Output modules, speech recognition and synthesis are also taken into account in our simulation as a reason for errors in the communication flow. However, in order to reflect human-machine dialogue reality, a simulated user always understands what the system says; hence no errors are introduced in the synthesis side. But on top of the vector representing the user DA, Automatic Speech Recognition (ASR) errors are simulated according to an error rate ER. With the probability $1 - ER$ the sentence is well recognised and $o_t^U = a_t^U$, and with the probability ER, the vector is corrupted with a noise vector n_t uniformly sampled on the domain set: $o_t^U = (a_t^U + n_t)/2$.

3.4. Application Simulator

In order to simulate the application behaviours and the inter-application ambiguities, a correspondence between applications and domains is introduced. In the simulation it is represented by means of a $n \times m$ matrix $C = \{c_{ij}\}$, where $c_{ij} = 1$ if the application α_i can manage domain δ_j and 0 otherwise. To model the domain similarity, we use a symmetric $m \times m$ matrix $D = \{d_{jk}\}$. $0 \leq d_{jk} = d_{kj} \leq 1$ represents the similarity between domains δ_j and δ_k.

The application scores are computed in the application simulator given observation o_t^U, matrices C and D as follows: first the domain similarity matrix D is multiplied by the input observation o_t^U, in order to compute the domains of interest, and then by the application-domain correspondence matrix C to score how intensely each application considers itself relevant with o_t^U. Finally, in order to add a variability on how the applications are understanding the input, a Gaussian noise $\mathcal{N}(0, \sigma^2)$ is added. This variability represents the I/O errors of the applications (due to a natural language understanding module). The output vector $sc_t = (sc_t^{\alpha_i})_{i \in [\![1,n]\!]}$ is therefore computed as follows:

$$sc_t = CDo_t^U + \left(\mathcal{N}_i(0, \sigma^2)\right)_{i \in [\![1,n]\!]}$$

4. THE DISPATCHER

The role of the Dispatcher is to choose, given the application scores $sc_t^{\alpha_i}$ and a dialogue history, the best action a_t^S to propagate to the user. A reinforcement learning approach is used to find the optimal strategy for this task.

4.1. Reinforcement Learning basics

The dialogue is formalised as an MDP $< \mathcal{S}, \mathcal{A}, P, R, \gamma >$ where: S is the state set, A the action set, $P : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$ the stochastic state transition function, $R : \mathcal{S} \times \mathcal{A} \rightarrow \mathbf{R}$ the stochastic reward function and $\gamma \in [0, 1)$ a discount factor penalising long-term rewards.

The Q-function is the expected value of taking an action a in a state s under a policy π. The optimal policy π^* maximises the Q-function and its optimal value is noted Q^*. It verifies the Bellman optimality equation:

$$Q^*(s,a) \quad = \quad \mathbb{E}[R(s) + \gamma \max_{a'} Q^*(P(s,a), a')]$$

Since $\gamma < 1$, a procedure called *value iteration* can be used to estimate Q^* [10]. The representation of the function can be done by using a table with an entry for each state-action pair. However, when the system state space is large or even continuous, like in dialogue systems, a parametric function [20] can be used as an approximation of the Q-function:

$$\hat{Q}_\theta(s,a) = \theta^t \phi(s,a)$$

where $\phi(s,a)$ is the p-dimensional feature vector and $\theta \in \mathbb{R}^p$ is the parameter vector and $\hat{Q}_\theta$ the approximate Q-function. The goal is to find a good approximation $\hat{Q}^*$ of Q^*, therefore to optimize θ. To do so, we use Fitted-Q Iteration algorithm [21, 22].

4.2. MDP modelling of the Dispatcher belief

State: The information represented in one state corresponds to the system's view of the user's intent. The retained state variables are: scores of applications, user request, history of the system proposal and grounding information.

We use a grounding model to report the consistency of the information exchanged by the different participants of a dialogue, here the user and the system about which application or domain is required. The grounding as described in [23] serves to resolve disagreement in a dialogue. A grounded action is the one which the user has approved. The various values of the grounding status are: *init*(α_i) initial state of the application α_i, *dnd*(α_i) when the application α_i has been denied by the user when it was proposed to her and *gnd*(a_i) means that the user agreed about a_i. The history of proposal is summarised in the array *lastTime*(α_i) which values are 0, if α_i was never selected, 1 if it was offered during the previous turn, and k if it has been offered k turns before.

To save computation time, we use a summarised version of the state information in order to use only the relevant features for the learning phase. As a result, the state representation $\phi(s)$ is composed of: scores of the 1st and 2nd-best scored application, grounding informations of the 1st and 2nd-best scored application (1 for *gnd*, -1 for *dnd*, 0 for *init*), history informations of the 1st and 2nd-best scored application, and turn count.

Subclass Name	Characteristics
App Family 1	All the applications belongs to exactly one domain and all the domains are disjoint.
App Family 2	All the applications belong to one unique domain.
App Family 3	Applications belong to up to 2 domains. But one of them is greedy, meaning that it belongs to all available domains. The elements of the D matrix are picked randomly.
App Family 4	Mix of App Family 1 and 3. Applications belong to up to 2 domains. But one of them is greedy. The domains of the system are disjoint (consequently C is the same as Family 3 and D as for Family 1).

Table 3. Subclasses of the application family

Actions: In practice, the action set (*i.e.* the system DAs) defined in Table 1 is too large. So, in the MDP only the acts are used, therefore $\mathcal{A}$ = {`offer`, `confExpl`, `askSelect`, `askRepeat`, `bye`} without their arguments. A heuristic function is defined manually to select the arguments of the selected act in order to obtain the full DA. Two heuristics are defined. The first one always select the best scored application as the argument of the DA; we call it *best heuristic* or simply *best*. The second heuristic proposes the last application successfully proposed that was not denied when it existed, we call it *last heuristic* or simply *last*.

Reward: The reward function is defined as follows:

$$\mathcal{R} = \begin{cases} -\lambda_{DL} & \text{in the intermediary turns} \\ -\lambda_{HU} & \text{if the user hangs up} \\ \lambda_{TC} & \text{when task is completed} \end{cases} \quad (1)$$

where t represents the duration of the dialogue, and λ_{DL}, λ_{HU}, and λ_{TC} are positive weights that respectively control the relative importance of the different key performance indicators: dialogue length, user hang-up and task completion.

5. EXPERIMENTS

5.1. Settings

Many configurations of parameters are possible for the simulator. Ideally, we should be able to define the different parameters from some initial sets of real data. However as such VPA system doesn't exist yet, none are available. After a sensitivity analysis, where the different variable influences have been studied, we make reasonable assumptions on the behaviours of the users and the applications. They make possible the definition of a first set of simulator parameters and run some proof-of-concept experiments. The parameters derived from these assumptions should be general enough to match the true

ones. We expect to demonstrate it after collecting data from a future system. We then identify and group into families some of the parameter values.

The **goal family**: we intend to simulate the diversity in user scenarios through three subclasses. The *app-goal* subclass consists in goals where the tasks are only about applications ($p_\tau = 1$). The *domain-goal* subclass corresponds to goals where the tasks are only about domains ($p_\tau = 0$). Then the *mixed-goal* subclass has a mixed type of tasks ($p_\tau = 0.5$). To generate average length goals, the parameter of the geometric distribution that determines the length of the goals p_G is set to 0.7 for all the subclasses.

The **application family**: it groups various configurations of application portfolios according to their behaviours. It is characterized by the values of the matrices C and D. Table 3 lists all the different sub-classes considered in our experiments. Of course this list is no exhaustive, but representative enough to give us bound values of how groups of applications can be situated in terms of the domains they address and how these domains may overlap or be confusing.

The **user family**: we categorize the users according to their patience and their agenda. To be more specific, we consider three profiles of users. The *patient* user is characterized by a high patience parameter. The *impatient* user is characterized by a low patience parameters. And an intermediary profile, *normal*, is also defined. The parameter controlling the length of the agenda is set identically for all the types of users, $p_{Ag} = 0.5$.

5.2. Experiment specifications

In the experiments, 10 applications and 10 domains are considered for the subclasses 1, 3 and 4 of the Application Families. The subclass 2 is composed of 10 applications and 1 domain. This number has been selected as representative of the normal set of applications that a user may use on a daily basis (on a smartphone for instance). Also, it allows to situate our work with respect to standard approaches where hardly more than 2-4 applications are considered[1] The noise vector added to the application score computation is generated with a variance of $\sigma^2 = 0.2$. And for sake of simplicity, in all the experiments only the *app-goal* subclass of the goal family has been evaluated.

In order to generate initial data and to test policies, we define two handcrafted policies. The *offer* policy always presents the best choice (in terms of application score) to the user, and never asks for clarification. The *random policy* selects randomly an action between: `offer`, `confExpl`, `askSelect` and `askRepeat` (a backoff mechanism is implemented if the selected action is not possible at some point).

[1]Obviously, these applications are to be specially designed for this purpose. this is also a huge burden in the system design, that we intend to greatly reduce here.

5.3. Results and discussions

Our aim is to find optimal strategies according to the different settings we defined above. The figures in this subsection show the average discounted sum of rewards and associated standard deviation over 50 parallel runs for policy learned using the Fitted-Q Iteration algorithm, with discount factor $\gamma = 0.9$.

With Fitted-Q Iteration, the policy is updated every 500 dialogues for a total of 2500 dialogues to ensure convergence. An ϵ-greedy policy is used with $\epsilon = \frac{1}{2j}$ where j is the iteration index. The learning starts using a *random policy* in order to collect data. Then with the collected dialogues, the θ parameters are computed and will be used for the upcoming dialogue collection until the next update.

The experimental results presented in Figure 2 show that the dialogue policy learned using Fitted-Q performs better than the handcrafted strategies for the subclasses 2, 3 and 4 with an ER of 0.1 and a patient user. Not surprisingly for the Family 1, where applications and domains are highly distinguishable, the simple *offer* policy stays above the others. Then we increase the noise level in the input: ASR error rate from 0.2 to 0.7 (the latter is not very realistic but it allows to strengthen the effects on the plot, and a normal, less patient, user). We observe in Figure 3 that the *offer* policy has a reduced average return, and that after several thousand sample dialogues the RL policy can beat it by about 10% relative.

On a regular basis it can also be observed from the figures that the trained strategy is sample efficient and needs only 1000 dialogues to outperform the handcrafted policies. Yet in this implementation, Fitted-Q Iteration has been used but more efficient solutions exist, as in [24, 25], and could even increase further the training speed.

6. CONCLUSION

In this paper, we introduce a novel approach for building a multi-application dialogue system based on a user-defined portfolio of applications covering several domains. This approach differs from the standard monolithic multi-domain systems, thanks to a new module, the Dispatcher, in charge of selecting the best application to answer a user request at anytime. It should make possible then to aggregate dialogue applications developed by any contributor and not only by the main system's owner. In this framework, the Dispatcher functioning is of paramount importance and its strategy should be learned from data and adapted online, to address the possible evolution of the portfolio.

To test our proposition, we designed a simulator with various parameters which tuning may fit real-life dialogue situations. These situations are related to user profiles (expertise, patience etc.), application and domain characteristics (domain confusion, application overlapping etc.), system

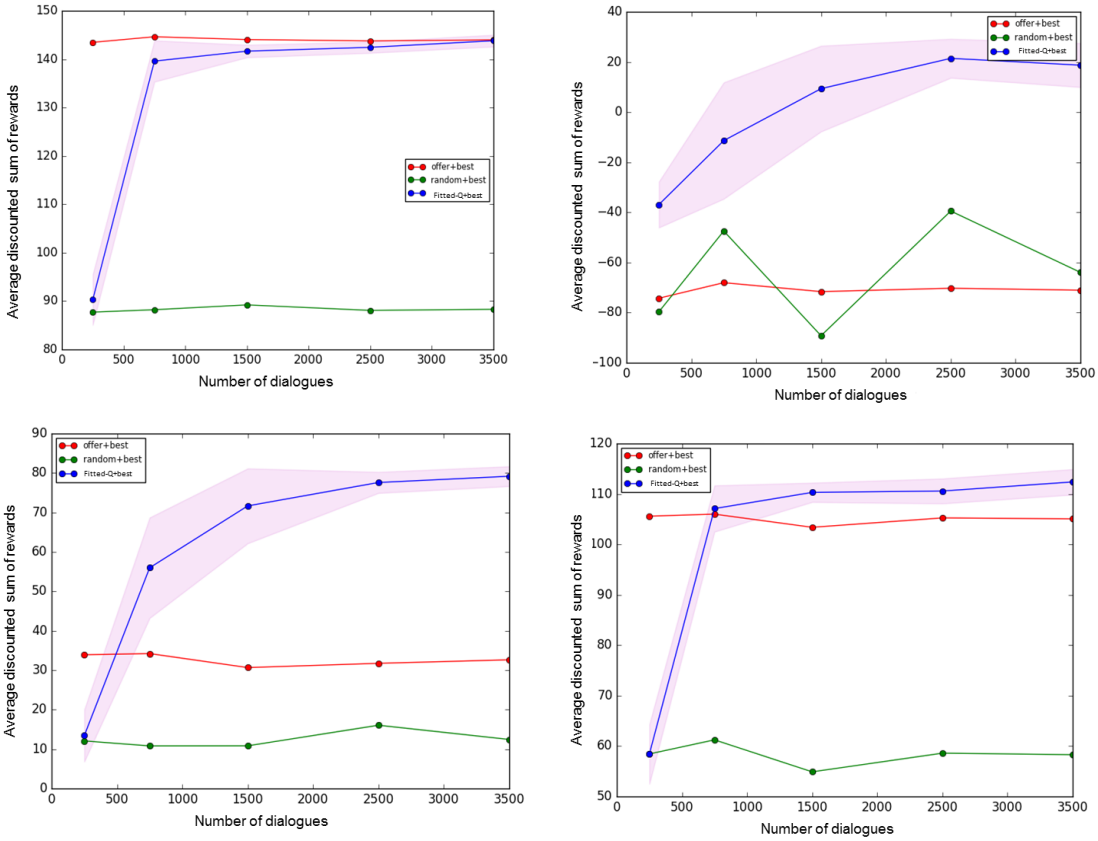

Fig. 2. All Application Families ($ER = 0.1$): 1 top left, 2 top right, 3 bottom left and 4 bottom right

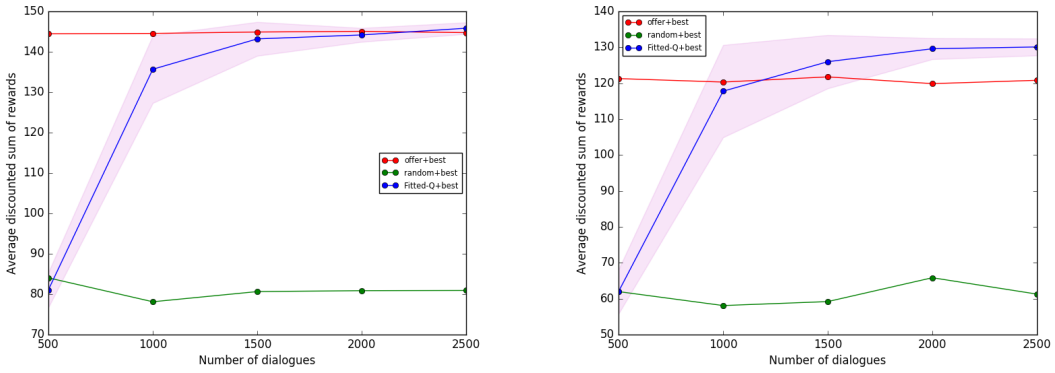

Fig. 3. Application Family 1 with ER of 0.2 (left) and 0.7 (right).

recognition and understanding errors. A first set of experiments shows that, except for situations in which applications are highly domain-dependent, reinforcement learning-based strategies outperform simple handcrafted ones. Furthermore, in all situations, the trained strategies are preferable when the noise level in the system's inputs is increased.

Many challenges remain to build a complete system based on this proposition. To name a few, out-of-domain handling, applications overestimating their scores, adaptation to a specific user [26], non stationary usages of sets of application and more generally co-adaptation [27, 28] belong to our concerns. To address most of them, our next step is to build a real system based on several available applications and to collect field data of users defining their own scenarios of usage and then incrementally add new applications.

7. REFERENCES

[1] Joaquin Planells, Lluıs-F Hurtado, Encarna Segarra, and Emilio Sanchis, "A multi-domain dialog system to integrate heterogeneous spoken dialog systems," in *Proceedings of the 13th Annual Conference of the International Speech Communication Association (Interspeech)*, 2013.

[2] Zhuoran Wang, Hongliang Chen, Guanchun Wang, Hao Tian, Hua Wu, and Haifeng Wang, "Policy learning for domain selection in an extensible multi-domain spoken dialogue system," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2014, pp. 57–67, Association for Computational Linguistics.

[3] Tatiana Ekeinhor-Komi, Hajar Falih, Christine Chardenon, Romain Laroche, and Fabrice Lefèvre, "Un assistant vocal personnalisable," in *Proceedings of the 21st Conférence sur le Traitement Automatique des Langues Naturelles (TALN)*, Marseille, France, July 2014, pp. 28–29, Association pour le Traitement Automatique des Langues.

[4] Bor-shen Lin, Hsin-ming Wang, and Lin-shan Lee, "A distributed architecture for cooperative spoken dialogue agents with coherent dialogue state and history," in *Proceeding of the 2nd IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*. Citeseer, 1999, vol. 99, p. 4.

[5] Mikio Nakano, Shun Sato, Kazunori Komatani, Kyoko Matsuyama, Kotaro Funakoshi, and Hiroshi G Okuno, "A two-stage domain selection framework for extensible multi-domain spoken dialogue systems," in *Proceedings of 12th Annual Meeting of the Special Interest Group on Discourse and Dialogue (Sigdial)*. Association for Computational Linguistics, 2011, pp. 18–29.

[6] Kazunori Komatani, Naoyuki Kanda, Mikio Nakano, Kazuhiro Nakadai, Hiroshi Tsujino, Tetsuya Ogata, and Hiroshi G Okuno, "Multi-domain spoken dialogue system with extensibility and robustness against speech recognition errors," in *Proceedings of the 10th Annual Meeting of the Special Interest Group on Discourse and Dialogue (Sigdial)*. Association for Computational Linguistics, 2009, pp. 9–17.

[7] Wei-Tek Hsu, Huei-Ming Wang, and Yi-Chun Lin, "The design of a multi-domain chinese dialogue system," in *Proceedings of the 3rd International Symposium on Chinese Spoken Language Processing (ISCSLP)*, 2002.

[8] Botond Pakucs, "Towards dynamic multi-domain dialogue processing.," in *Proceedings of the 3rd Annual Conference of the International Speech Communication Association (Interspeech)*, 2003.

[9] Satoshi Ikeda, Kazunori Komatani, Tetsuya Ogata, and Hiroshi G. Okuno, "Integrating topic estimation and dialogue history for domain selection in multi-domain spoken dialogue systems," in *New Frontiers in Applied Artificial Intelligence*, pp. 294–304. Springer, 2008.

[10] Richard S Sutton and Andrew G Barto, *Reinforcement learning: An introduction*, vol. 1, MIT press Cambridge, 1998.

[11] Esther Levin, Roberto Pieraccini, and Wieland Eckert, "Learning dialogue strategies within the markov decision process framework," in *Proceeding of the 1st IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*. IEEE, 1997, pp. 72–79.

[12] Olivier Lemon and Olivier Pietquin, "Machine learning for spoken dialogue systems," in *Proceedings of the 7th Annual Conference of the International Speech Communication Association (Interspeech)*, 2007, pp. 2685–2688.

[13] Romain Laroche, Ghislain Putois, Philippe Bretier, and Bernadette Bouchon-Meunier, "Hybridisation of expertise and reinforcement learning in dialogue systems," in *Proceedings of the 10th Annual Conference of the International Speech Communication Association (Interspeech), Special Session: Machine Learning for Adaptivity in Spoken Dialogue*, Brighton (United Knigdom), 2009.

[14] Wieland Eckert, Esther Levin, and Roberto Pieraccini, "User modeling for spoken dialogue system evaluation," in *Procceding of the 1st IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*. IEEE, 1997, pp. 80–87.

[15] Kallirroi Georgila, James Henderson, and Oliver Lemon, "Learning user simulations for information state update dialogue systems.," in *Proceedings of the 5th Annual Conference of the International Speech Communication Association (Interspeech)*, 2005, pp. 893–896.

[16] Jost Schatzmann, Blaise Thomson, Karl Weilhammer, Hui Ye, and Steve Young, "Agenda-based user simulation for bootstrapping a pomdp dialogue system," in *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Companion Volume, Short Papers*. Association for Computational Linguistics, 2007, pp. 149–152.

[17] Olivier Pietquin and Thierry Dutoit, "A probabilistic framework for dialog simulation and optimal strategy learning," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 14, no. 2, pp. 589–599, 2006.

[18] Hatim Khouzaimi, Romain Laroche, and Fabrice Lefèvre, "Reinforcement learning for turn-taking management in incremental spoken dialogue systems," .

[19] Steve Young, Milica Gašić, Simon Keizer, François Mairesse, Jost Schatzmann, Blaise Thomson, and Kai Yu, "The hidden information state model: A practical framework for pomdp-based spoken dialogue management," *Computer Speech & Language*, vol. 24, no. 2, pp. 150–174, 2010.

[20] Richard Bellman and Stuart Dreyfus, "Functional approximations and dynamic programming," *Mathematical Tables and Other Aids to Computation*, vol. 13, no. 68, pp. 247–251, 1959.

[21] Damien Ernst, Pierre Geurts, and Louis Wehenkel, "Tree-based batch mode reinforcement learning," *Journal of Machine Learning Research*, pp. 503–556, 2005.

[22] Senthilkumar Chandramohan, Matthieu Geist, and Olivier Pietquin, "Optimizing spoken dialogue management with fitted value iteration.," in *Proceedings of the 10th Annual Conference of the International Speech Communication Association (Interspeech)*, 2010, pp. 86–89.

[23] Susan Elise Brennan, *Seeking and providing evidence for mutual understanding*, Ph.D. thesis, Stanford University, 1990.

[24] Lucie Daubigney, Matthieu Geist, Senthilkumar Chandramohan, and Olivier Pietquin, "A comprehensive reinforcement learning framework for dialogue management optimization," *Selected Topics in Signal Processing, IEEE Journal of*, vol. 6, no. 8, pp. 891–902, 2012.

[25] Emmanuel Ferreira and Fabrice Lefèvre, "Reinforcement-learning based dialogue system for human–robot interactions with socially-inspired rewards," *Computer Speech & Language*, vol. 34, no. 1, pp. 256–274, 2015.

[26] Aude Genevay and Romain Laroche, "Transfer learning for user adaptation in spoken dialogue systems," in *Proceedings of the 15th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*. International Foundation for Autonomous Agents and Multiagent Systems, 2016.

[27] Senthilkumar Chandramohan, Matthieu Geist, Fabrice Lefèvre, and Olivier Pietquin, "Co-adaptation in spoken dialogue systems," in *Natural Interaction with Robots, Knowbots and Smartphones*, pp. 343–353. Springer, 2014.

[28] Merwan Barlier, Julien Perolat, Romain Laroche, and Olivier Pietquin, "Human-machine dialogue as a stochastic game," in *Proceedings of the 16th Annual Meeting of the Special Interest Group on Discourse and Dialogue (Sigdial)*, 2015.