User Simulation in Dialog Systems using Inverse Reinforcement Learning

Shobhit Chaurasia
Department of Computer Science
UTEID: sc52987

1. ABSTRACT

In Spoken Dialog Systems (SDS), an agent interacts with a human using natural-language to complete a task. Agents in SDS — often called Dialog Manager (DM) — are usually trained using Reinforcement Learning (RL). Since RL algorithms are typically data hungry, DMs are trained against simulations of human users. Traditionally, user-simulators have been built directly using the statistical patterns mined from a dialog corpus, or through supervised learning approaches. In this paper, Apprenticeship Learning is used to build user-simulators from sample dialog conversations using Inverse Reinforcement Learning.

2. INTRODUCTION

SDS are commonly used in restricted domains, such as tourist-information domain [11] and flight-booking domain [19], where the system requests a pre-defined set of information from the user and generates a response based on the acquired knowledge. Since dialog in inherently sequential, the task of training a DM is often posed as an RL problem. Training an RL agent requires large number of interactions with the environment; in the domain of SDS, this involves interaction with humans, which is an expensive task. Moreover, learning policies for a DM — just like learning in other RL domains in which interaction with the environment has high-stakes — poses a "chicken-and-the-egg" problem: to learn a policy for DM, interaction with users is required; however, this initial interaction during the learning phase itself requires a decent policy for the DM. A common way to mitigate these issues is to build a simulated user with which the DM can interact large number of times to learn either the final policy or, at least, a decent one to bootstrap its learning-by-interaction phase. At a time when chat-bots are becoming prevalent and gaining traction among users and companies alike, building interactive agents is an important problem, and training them through simulations of users might be a feasible approach before the "beta" versions of chat-bots are released to public for fine-tuning and testing.

Traditionally, user simulations have been built either by replicating statistics of user-behavior in a dialog corpus or by posing it as a supervised learning problem. Such user simulations, however, are not adaptive: unlike humans who adapt their behavior in response to the way the conversation is progressing, such simulations follow a fixed pattern learned from the dialog corpus they were trained on. Since statistics of behavioral similarity between real users and user-simulators is often used as an evaluation metric for the quality of user simulation, it is questionable whether they should be built using the same metric in the first place.

Majority of SDS proposed in recent literature are based on RL. While they pose the problem of building a DM as an RL problem, simulating a user is a symmetric problem, with the roles of the DM and user interchanged. In this paper, user-simulator for SDS is built using Inverse Reinforcement Learning (IRL). Given samples from the expert (real-users) dialog policy, the task is to simulate a user that mimics the expert. Precisely, the task is to first learn the unknown reward function underlying the expert's style of conversation with DM, and then use it to learn a similar policy.

The idea is based on [3]. The goal of this study is to re-implement their system and evaluate the viability of IRL for building user-simulators in the domain of SDS.

This paper is structured as follows: After a brief survey of existing user-simulation techniques in Section 3, MDP and IRL are defined in Sections 4 and 5 respectively. Details of the main approach to user simulation using IRL are presented in Section 6. This is followed by experiments and discussion of results in Sections 7 and 8 respectively. After summarizing the study in Section 9, the paper ends with directions for future work noted in Section 10.

3. RELATED WORK

User simulation can be performed at varying granularity: acoustic-level, word-level, and intention-level [17]. Among these, the intention-level simulation — which simulates users at an abstract level of the intentions behind their utterances through speech-acts, such as `provide_origin_city`, `provide_departure_date`, and `confirm_flight` in a flight-booking domain – is the most common since the other two can be built as separate components to be plugged into the intention-level simulation. Traditionally, the intention-level user simulation has been approached with methods that attempt to replicate user-utterance statistics gleaned from dialog corpora [13, 8], those that rely on supervised machine learning models [10, 7], and those that model user agenda — and more recently, goals — for conversations [16, 9].

An interesting approach to user simulation is proposed in [3] where modeling of user behavior is posed as an RL problem. The framework of Markov Decision Process (MDP) is used for the task of user modeling. MDP, however, requires the specification of a reward function R which characterizes the goal of the task; the optimal policy π for the task is defined as one that maximizes the expected cumulative reward under π. In dialog systems, specification of a reward function characterizing user behavior is non-trivial and of-

ten unclear. Although there have been attempts at manually defining reward functions, such as those in the PARADISE framework [18], this approach is susceptible to bias being introduced by the designer in the user models. To do away with the need for a handcrafted reward function, [3] uses IRL to automatically uncover the hidden reward function — that the user might be trying to optimize — given sample dialog conversations. With a reward function at their disposal, a user-simulator is trained using RL algorithms to obtain a policy — i.e., user strategy — that is optimal with respect to the unknown reward function of the expert (user).

IRL was formalized in [12] as the problem of discovering the hidden reward function, R^* underlying a policy that is optimal with respect to R^*. Often, as is the case with all dialog corpora used to build SDS, instead of the optimal policy, only a set of trajectories sampled from the optimal policy are available.

The task of teaching an agent to perform a task through sample expert demonstrations is called apprenticeship learning, imitation learning, or learning from demonstration. Previous research in the domain of apprenticeship learning has focused on imitating the expert's behavior by posing it as supervised learning problem, learning a mapping from states to actions directly [14, 15, 6, 2]. However, since reward function is the most succinct representation of a task, as is well accepted in the RL domain, [1] uses IRL to use reward function as a proxy for imitating expert behavior.

User modeling in [3] uses the IRL-based apprenticeship learning proposed in [1]. A particular advantage of treating user-simulator as an RL agent and extracting reward function, or related metrics such as feature expectations [1], enroute to learning a policy that imitates user behavior is that it lends itself to other tasks, such as user-behavior clustering [4] and co-adaptation of dialog system and user simulation to facilitate evolving dialog strategies over the course of conversations [5], which can further improve the quality of user simulation.

4. MARKOV DECISION PROCESS

RL problems are formulated as Markov Decision Processes (MDPs) which are characterized by the tuple $\{S, A, T, R, \gamma\}$, where S is the state-space of the agent; A is its action-space; the transition function (probability distribution over next-states given current state and action) of the environment is defined by $T : (S, A) \rightarrow \mathbb{P}(S')$; $R : (S, A) \rightarrow \mathbb{R}$ characterizes the reward that the agent receives after a transition; and γ, the discount factor, governs the ratio of importance given to immediate rewards versus future rewards.

In the domain of SDS, user acts as the environment for the DM. In general, user's transition function (his state of mind and his future utterances) and user's reward function (a representation of his ultimate goal) are unknown. Users can be seen as following a policy that maximizes their reward function. By treating user-simulator as a RL agent that can mimic real-users, the task is, then, to uncover the hidden reward function and use off-policy methods to learn the real-users' policy.

5. INVERSE REINFORCEMENT LEARNING

Given sample episodes generated by following an *expert* policy, the problem of discovering the hidden reward function which makes the expert-policy optimal is called IRL [12]. The actual policy is not available.

Since the entity of interest is usually a policy close to the expert-policy (derived from the learned reward function), [1] poses imitation learning as an IRL problem. The proposed method assumes that the expert, during demonstrations, is trying to maximize an unknown reward function expressible as a linear combination of known features.

$$R(s, a; \theta) = \theta^\top \phi(s, a) \tag{1}$$

where $\phi : (S \times A) \rightarrow [0, 1]^k \in \mathbb{R}^k$ is the known feature vector for each state-action pair, and the parameters θ are such that $\|\theta\|_2 = 1$. Define *feature expectation* of policy π as:

$$\mu^\pi(s_0, a) = \mathbb{E}\left[\sum_{i=0}^{k} \gamma^i \theta^\top \phi(s_0, a)\right] \tag{2}$$

where s_0 is the fixed start-state, and the action a in s_0 is sampled from π. The state-value function, Q^π, for policy π can be written in terms of its feature expectation:

$$Q^\pi(s, a) = \theta^\top \mu^\pi(s, a) \tag{3}$$

Since value functions quantify the quality of a policy, feature expectation is, thus, a succinct representation of a policy. The method proposed in [1] progressively finds policies whose feature expectations are close to that of the expert-policy.

The goal of finding a policy that performs almost as well as the expert-policy — performance being measured with respect to the unknown reward function — boils down to an SVM-like optimization problem. Details of the algorithm are given in [1]. It converges to a policy that attains performance close to that of the expert. However, unlike traditional IRL algorithms, it may not recover the expert's reward function; this is not a deal-breaker since the entity of interest is a near-optimal policy.

The algorithm needs an estimate, $\hat{\mu_E}$, of the feature expectation of the expert policy. Estimates for feature expectations for a policy π can be calculated from the sample trajectories (episodes) obtained drawn from that policy.

$$\hat{\mu}(s_0, a) = \frac{1}{n} \sum_{i=0}^{n} \sum_{t=0}^{l_i} \gamma^t \phi\left(s_t^{(i)}, a_t^{(i)}\right) \tag{4}$$

where n is the number of episodes, and l_i is the length of i^{th} episode.

6. USER SIMULATION USING IRL

Just like DM is cast an RL agent when its policy needs to be learned, the user-simulator is cast as an RL agent the same way. The MDP for user-simulator is defined by $(S_{sim}, A_{sim}, T_{sim}, R_{sim}, \gamma)$. S_{sim} and A_{sim} are domain-specific state-action pairs for user-simulator (see Section 7 for details); R_{sim} is the unknown reward function that we wish to extract through IRL. Access to the transition function — model of the environment — is not required since we can use off-policy MDP solvers such as SARSA. The DM acts as the environment for the user-simulator.

The algorithm returns a set of policies $\Pi = \{\pi^0, \ldots, \pi^m\}$, along with a set of the associated feature expectations $\Omega = \{\mu^0, \ldots, \mu^m\}$, for the user-simulator learned during its execution. Additionally, it returns an estimate of their distances

to the expert-policy — approximated by the distance between the their feature expectations and that of the expert-policy – feature expectation of expert policy is estimated by 4. The policy which elicits user-utterance pattern closest to that of the expert can either be picked manually from this set or found by solving a Quadratic Programming (QP) problem that finds the policy closest to the expert policy — distance being measured in terms of feature expectations — that lies in the convex closure of the policies in Π:

$$\min \|\mu_E - \mu\|_2$$
$$\text{such that} \quad \mu = \sum_i \lambda_i \mu^{(i)}, \quad \lambda_i \geq 0, \quad \sum_i \lambda_i = 1 \qquad (5)$$

At each iteration of the algorithm, a new set of weights θ are learned. The reward function R corresponding to these set of weights is used by an MDP solver (such as SARSA) to learn an optimal policy under R. Since every stationary environment has a deterministic optimal policy, the policy returned by the MDP solver is always deterministic. Hence, the set of policies Π maintained by IRL algorithm consists only of deterministic policies. This has the implication that the user-simulator will always take the same action whenever it lands in a particular state. This deviates from how humans behave: their behavior, and hence the policies they are, perhaps unknowingly, following are stochastic. This is not a failure of IRL, though. To understand why, it is important to note that Π contains at least one policy "whose performance under R^* is at least as good as the expert's performance minus ϵ," [1] for some small $\epsilon > 0$. IRL does not strictly assume that the policy that the expert is following is optimal with respect to the underlying reward function R^*. Hence, the aim of IRL in itself is not to mimic the expert's policy; rather, it strives to learn a policy that is close to that of the expert in terms of its performance, perhaps even better. When the expert-policy is sub-optimal, it is possible that one of the policies in Π are better than the expert-policy. Therefore, it makes sense that the policies returned by IRL are deterministic. However, since our aim here is not to build a user-simulator which outperforms users in terms of efficiency of dialog, but rather to mimic the (potentially sub-optimal) style of users, an additional operation on top of IRL is needed.

6.1 Mixing Learned Policies

To obtain a stochastic policy for user-simulator using the deterministic policies Π learned by IRL, two strategies are tested: mixing policies using feature expectations, and discovering mixed policies using QP.

6.1.1 Mixing Policies using Feature Expectations

The policies in Π can be mixed to obtain a stochastic policy by stochastically selecting policy π_i with probability p_i before each dialog session (i.e., before each episode). The mixing weights p_i can be determined from the distance, d_i, of the feature expectation, μ_i, of each policy from μ_E. Specifically, we can use *softmax* of the distances as mixing weights:

$$p_i = \frac{e^{-d_i}}{\sum_i e^{-d_i}} \qquad (6)$$

where $d_i = \|\mu_i - \mu_E\|_2$

The hope is that policies learned during IRL include those that characterize the sub-optimal actions of users in certain states, and by mixing these policies in proportion to their distances to the expert-policy, expert's behavior can be mimicked.

6.1.2 Mixing Policies using QP

The solution of QP in Eq. 5 is a point whose feature expectation is a convex combination of the feature expectations of the learned policies, $\lambda_i \forall i \in \{1, \ldots, m\}$. It is shown in [1] that a policy π_d whose feature expectation is a convex combination of feature expectations of a set $S = \{\pi_1, \ldots, \pi_m\}$ of policies can be obtained by picking policy π_i stochastically before each episode with probability λ_i and executing it. The policy π_d, thus, is a stochastic policy whose feature expectation is closest to that of the expert within the convex hull enclosing all other policies. Therefore, it is a promising candidate for a stochastic policy mimicking user's behavior. This strategy is similar to the previous one, the difference being the way the mixing weights are determined.

7. EXPERIMENTS

Ideally, experiments should be conducted on a real-world dialog corpus which has intention-level tags, such as the COMMUNICATOR corpus [19]. However, due to its non-availability, and since the existence of a real-user is not crucial to evaluation of the methods described above, experiments are performed on a simple domain of Restaurant-Information system used in [3]. A hand-crafted "expert user-simulation" interacts with the DM to generate a simulated dialog corpora. This simulated corpus is used as a replacement for a real-world dialog corpus. The DM in Restaurant-Information domain requests information about *location*, *price-range*, and *cuisine* from the users. From the perspective of the DM, it can be viewed as a slot-filling task. The state and action spaces of the DM and user-simulator are described in the next section, followed by a discussion about the experiments and their results.

7.1 State and Action Spaces of DM

The DM requests information about the three slots from the user and confirms the information it receives. The status of the three slots — `Empty`, `Requested`, and `Confirmed` — defines the state-space of the DM. Its actions include:

- an action to start the dialog with a greeting (`Greet`);
- three actions for requesting slots (`Ask-Slot`), one for each slot;
- three actions for explicitly confirming slots (`Explicit-Confirm`), one for each slot;
- six actions corresponding to implicit confirmation of a slot along with a request for another slot (`Implicit-Confirm`); and
- an action to terminate the dialog session (`Close`)

Since the focus here is not on learning good policies for DM, a simple hand-crafted policy is used for DM. The policy requests and confirms the three slots one after the other.

7.2 State and Action Spaces of User-Simulator

The user-simulator provides information about the three slots to the DM and reaffirms or negates the information that DM wishes to confirm. Its state is characterized by the

status of the three slots: `Empty` if the slot hasn't yet been provided by it, `Provided` if the slot was provided by it in the past, and `Confirmed` if the slot has already been confirmed by it. Additionally, the previous action of the DM is also part of its state-space: the user-simulator's utterances, just like real-users', are a response to DM's requests. Its actions include:

- a `Silent` action where it stays silent;
- three actions for providing one slot at a time (`One-Slot`), one for each slot;
- an action for providing all slots in one go (`All-Slots`);
- three actions to provide positive confirmation for the slots (`Confirm`), one for each slot;
- three actions to negate confirmation request for the slots (`Negate`), one for each slot; and
- an action to terminate the dialog session (`Close`)

The hand-crafted policy used for "expert-user-simulator" is described in Table 1. The task is, then, to learn to imitate the behavior elicited by the hand-crafted policy for the expert-user-simulator.

DM Action	User-Simulator Actions
`Greet`	`Silent` (0.5); `All-Slots` (0.5)
`Ask-Slot`	`One-Slot` (0.8); `All-Slots` (0.2)
`Explicit-Confirm`	`Confirm` (1.0)
`Implicit-Confirm`	`One-Slot` (0.5); `Negate` (0.5)
`Close`	`Close` (1.0)

Table 1: Hand-crafted policy for "expert-user-simulation."

7.3 Algorithm Details

SARSA is used as the MDP solver in the IRL algorithm, with learning rate and degree of exploration decayed every episode. Since the state and action spaces are small, 100 learning episodes in SARSA suffice. Feature expectations are estimated by running 10,000 dialog sessions between user-simulator and the hand-crafted DM.

7.4 Evaluation of User Simulation

IRL algorithm is run for 300 steps. The set of policies returned by IRL algorithm for user-simulator are used to learn three different types of user-simulators: *Single-best-simulation*, *Softmax-simulation*, and *QP-simulation*.

- *Single-best-simulation* corresponds to the best policy among the pool of policies returned by IRL. This is a deterministic policy.
- *Softmax-simulation* corresponds to the policy obtained by mixing the individual policies as described in Section 6.1.1
- *QP-simulation* corresponds to the policy obtained by solution to QP in 5 as described in Section 6.1.2

Tables 2, 3, and 4 describe the preferences of the three different user-simulators following different DM actions. Fig. 1 represents the user behavior learned by different simulators on a per-episode basis: it plots the frequencies of different user-simulator actions per dialog session.

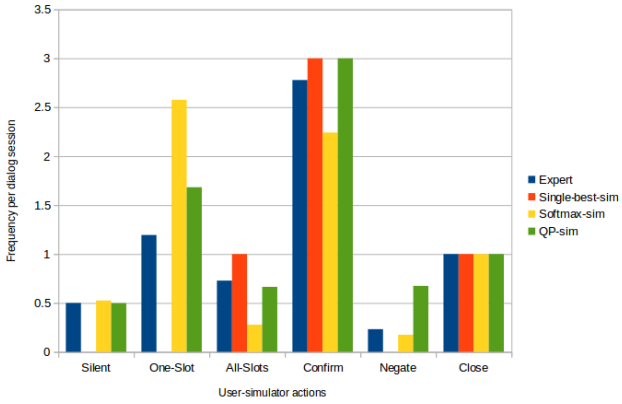

Figure 1: Frequency of user-simulator actions per dialog session for different simulators.

DM Action	User-Simulator Actions
`Greet`	`All-slots` (1.0)
`Ask-Slot`	`One-Slot` (1.0)
`Explicit-Confirm`	`Confirm` (1.0)
`Implicit-Confirm`	`One-Slot` (1.0)
`Close`	`Close` (1.0)

Table 2: Learned policy corresponding to *Single-best-simulation*

DM Action	User-Simulator Actions
`Greet`	`Silent` (0.53); `One-Slot` (0.43); `All-Slots` (0.04)
`Ask-Slot`	`One-Slot` (0.84); `All-Slots` (0.16)
`Explicit-Confirm`	`Confirm` (1.0)
`Implicit-Confirm`	`One-Slot` (0.84); `Confirm` (0.01); `Negate` (0.15)
`Close`	`Close` (1.0)

Table 3: Learned policy corresponding to *Softmax-simulation*

8. DISCUSSION

The policy for *Single-best-simulation* is the best individual policy returned by the IRL algorithm. IRL algorithm, therefore, succeeds at learning a policy that is optimal under the hidden reward function — completing a dialog session in fewest possible turns — that the user is trying to maximize unsuccessfully. However, for the task at hand (i.e., learning

DM Action	User-Simulator Actions
`Greet`	`Silent` (0.43); `All-Slots` (0.57)
`Ask-Slot`	`One-Slot` (0.93); `All-Slots` (0.07)
`Explicit-Confirm`	`Confirm` (1.0)
`Implicit-Confirm`	`One-Slot` (1.0)
`Close`	`Close` (1.0)

Table 4: Learned policy corresponding to *QP-simulation*

policies that mimic user behavior), this is not an acceptable user-simulator.

Both *Softmax-simulation* and *QP-simulation* learn policies that are stochastic and more representative of the expert user's policy than *Single-best-simulation*. Since *Softmax-simulation* mixes all the policies returned by IRL according to their distances from μ_E, it often ends up taking actions that have zero probability in expert-user's policy. Although *QP-simulation* takes the same approach, the policy learned by it is additionally guaranteed to be closest to expert-policy within the convex hull of learned policies. At least in theory, this makes it better than *Softmax-simulation*. Superiority of *QP-simulation* is not immediately clear from the description of the learned policies, but is evident from Fig. 1 which compares the learned behaviors at the level of individual episodes.

9. CONCLUSION

This study presents IRL as a viable approach to build user simulations. Deterministic policies returned by the IRL algorithm can be mixed in different ways to build stochastic user-simulators that, like real-users, might not be optimal with respect to the dialog strategy, but mimic users' behavior well.

10. FUTURE WORK

Since user-simulators are built for the purpose of training DM, a better evaluation metric for assessing the quality of user-simulators would be to compare policies of DMs that are trained by interaction with user-simulators.

Further, as proposed in [5], an interesting extension of this approach would be to train DM and user-simulator in an alternating fashion to allow them to co-adapt to each others' policies, as opposed to keeping one fixed while training the other.

11. REFERENCES

[1] Abbeel, P., and Ng, A. Y. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning* (2004), ACM, p. 1.

[2] Argall, B. D., Chernova, S., Veloso, M., and Browning, B. A survey of robot learning from demonstration. *Robotics and autonomous systems 57*, 5 (2009), 469–483.

[3] Chandramohan, S., Geist, M., Lefevre, F., and Pietquin, O. User simulation in dialogue systems using inverse reinforcement learning. In *Interspeech 2011* (2011), pp. 1025–1028.

[4] Chandramohan, S., Geist, M., Lefevre, F., and Pietquin, O. Clustering behaviors of spoken dialogue systems users. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2012), IEEE, pp. 4981–4984.

[5] Chandramohan, S., Geist, M., Lefevre, F., and Pietquin, O. Co-adaptation in spoken dialogue systems. In *Natural Interaction with Robots, Knowbots and Smartphones*. Springer, 2014, pp. 343–353.

[6] Chernova, S., and Veloso, M. Interactive policy learning through confidence-based autonomy. *Journal of Artificial Intelligence Research 34*, 1 (2009), 1.

[7] Cuayáhuitl, H., Renals, S., Lemon, O., and Shimodaira, H. Human-computer dialogue simulation using hidden markov models. In *IEEE Workshop on Automatic Speech Recognition and Understanding, 2005.* (2005), IEEE, pp. 290–295.

[8] Eckert, W., Levin, E., and Pieraccini, R. User modeling for spoken dialogue system evaluation. In *Automatic Speech Recognition and Understanding, 1997. Proceedings., 1997 IEEE Workshop on* (1997), IEEE, pp. 80–87.

[9] Eshky, A., Allison, B., and Steedman, M. Generative goal-driven user simulation for dialog management. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning* (2012), Association for Computational Linguistics, pp. 71–81.

[10] Kallirroi, G., Henderson, J., and Lemon, O. Learning user simulations for information state update dialogue systems. In *Interspeech* (2005), pp. 893–896.

[11] Lemon, O., Georgila, K., Henderson, J., and Stuttle, M. An isu dialogue system exhibiting reinforcement learning of dialogue policies: generic slot-filling in the talk in-car system. In *Proceedings of the Eleventh Conference of the European Chapter of the Association for Computational Linguistics: Posters & Demonstrations* (2006), Association for Computational Linguistics, pp. 119–122.

[12] Ng, A. Y., and Russell, S. J. Algorithms for inverse reinforcement learning. In *Proceedings of the Seventeenth International Conference on Machine Learning* (2000), Morgan Kaufmann Publishers Inc., pp. 663–670.

[13] Pietquin, O., and Dutoit, T. A probabilistic framework for dialog simulation and optimal strategy learning. *IEEE Transactions on Audio, Speech, and Language Processing 14*, 2 (2006), 589–599.

[14] Ross, S., , and Bagnell, D. Efficient reductions for imitation learning. *AISTATS* (2011).

[15] Ross, S., Gordon, G. J., , and Bagnell, D. A reduction of imitation learning and structured prediction to no-regret online learning. *AISTATS 1*, 2 (2011).

[16] Schatzmann, J., Thomson, B., Weilhammer, K., Ye, H., and Young, S. 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* (2007), Association

for Computational Linguistics, pp. 149–152.

[17] Schatzmann, J., Weilhammer, K., Stuttle, M., and Young, S. A survey of statistical user simulation techniques for reinforcement-learning of dialogue management strategies. *The knowledge engineering review 21*, 02 (2006), 97–126.

[18] Walker, M. A., Litman, D. J., Kamm, C. A., and Abella, A. Paradise: A framework for evaluating spoken dialogue agents. In *Proceedings of the eighth conference on European chapter of the Association for Computational Linguistics* (1997), Association for Computational Linguistics, pp. 271–280.

[19] Walker, M. A., Passonneau, R., and Boland, J. E. Quantitative and qualitative evaluation of darpa communicator spoken dialogue systems. In *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics* (2001), Association for Computational Linguistics, pp. 515–522.