Automatic learning of dialogue strategy using dialogue simulation and reinforcement learning

Konrad Scheffler
Department of Engineering
Cambridge University, Cambridge, UK
khs22@cam.ac.uk

Steve Young
Department of Engineering
Cambridge University, Cambridge, UK
sjy@cam.ac.uk

ABSTRACT

This paper describes a method for automatic design of human-computer dialogue strategies by means of reinforcement learning, using a dialogue simulation tool to model the user behaviour and system recognition performance. To the authors' knowledge this is the first application of a detailed simulation tool to this problem. The simulation tool is trained on a corpus of real user data. Compared to direct state transition modelling, it has the major advantage that different state space representations can be studied without collecting more training data.

We applied Q-learning with eligibility traces to obtain policies for a telephone-based cinema information system, comparing the effect of different state space representations and evaluation functions. The policies outperformed handcrafted policies that operated in the same restricted state space, and gave performance similar to the original design that had been through several iterations of manual refinement.

1. INTRODUCTION

In automatic design of dialogue strategy, a finite state representation of the dialogue is used so that the dialogue manager can be specified as an agent to be trained using machine learning techniques. We follow the approach [2, 17] of describing dialogue as a Markov decision process (MDP).[1] The dialogue strategy to be designed is then an MDP policy, which can be optimised by means of reinforcement learning algorithms [12].

[1]The presence of imperfect recognition means that the dialogue state is not fully observable and the dialogue would be better described as a partially observable Markov decision process (POMDP) [6, 18]. However, the computational complexity involved in learning policies for POMDPs makes this approach problematic. We argue that the benefits of having a tractable computational framework outweigh the slightly inferior modelling accuracy of MDPs.

.

There are two types of dialogue model that can be used for this purpose. The first [15, 14, 9, 10, 4, 11, 6] is a model of dialogue state transitions, which can be estimated directly from a corpus in which these transitions have been logged. The model is then used with a model-based reinforcement learning algorithm such as dynamic programming to find the optimal policy.

An alternative approach is to generate training episodes by simulation so that the optimal policy can be found using a simulation-based reinforcement learning algorithm such as Q-learning [3, 1]. An advantage of this approach is that the system state representation need not be fixed: if the chosen state space proves problematic during training, it can be changed and the system retrained without having to change the data set, data transcriptions or user model. However, the lack of detailed user models meant that this method could so far only be applied to very simple dialogues.

This paper describes how a quantitative, data driven model[8] of the user behaviour and system recognition performance can be used to enable complex strategies to be learnt in a real dialogue system.

2. APPLICATION TO A REAL DIALOGUE SYSTEM

The proposed approach to dialogue design was evaluated using an application which was originally developed by a team of dialogue design engineers and refined via user trials. The application provides cinema information via the telephone and it was used to collect a data corpus on which the user and error models were trained. Evaluation of the simulation system has been reported previously in [8].

2.1 User and error models

In the cinema information application, the user is able to access information via four transaction types: *film listings*, *film distribution*, *film times* and *cinema address*. It can be treated as a form-filling dialogue, with a transaction being described by filling in a subset of the fields *type*, *day*, *film* and *cinema*. Recognition rates and user response patterns, both of which are modelled by the dialogue simulation system, differ for the different fields.

The user model is based on the assumption that the user acts in a consistent and goal directed fashion, while varying his/her actions in a probabilistic way. The internal user state is modelled around a goal structure such as that in Table 1, and comprises the following variables

Goal field	Value	Status
Type:	FILMLIST	specified
Film:	NA	NA
Cinema:	CINEMANAME	urgent
Day:	DAYVALUE	pending

Table 1: Example of a user goal and associated status variables in the cinema application.

that are updated throughout the course of the dialogue:

- The current contents of the user goal
- Status variables associated with the goal fields.
- Secondary goal structures representing the user's beliefs on the current system goal and the newly completed goal (if any).
- Application specific variables describing items such as the type (eg. direct question) of the previous prompt.

The user and error models also use parameters estimated from the training data which quantify the frequencies of different user goal initialisations, how often particular types of details will be specified in mixed initiative situations, how likely each possible insertion, deletion and substitution error is to occur during recognition, and what the distribution of confidence levels is for both correct and incorrect recognitions. Training of these parameters is described in [8, 7].

The models are generated automatically from a dialogue task specification and the tagged syntaxes used by the recognition / understanding subsystem. They can then be used to generate intention level versions of responses to system utterances, governed by the model probabilities as well as the current user state. By interfacing them with an implementation of the system, complete dialogues can be simulated.

2.2 Action set

Some decisions that need to be taken in the course of the dialogue, such as deciding which slot should be in focus at a given moment, or which syntax to use after deciding to do a system initiative query on the current slot, are easy to design and can be implemented by hand. It is therefore only necessary to define a restricted set of actions to make available to the learning system. These actions determine the extent of the strategy learning problem, and the amount of freedom available in choosing between different policies. The action set chosen for the cinema application is summarised in Table 2. It allows for automatic design of the choice between mixed and system initiative and the confirmation strategy (a choice between explicit confirmation, implicit confirmation, and delaying confirmation until later).

2.3 Cost functions

Rather than enter the debate on which factors are important in dialogue system evaluation, we choose two easily measurable objective quantities: average transaction length in number of turns and average transaction success rate. These are then used as the basis for the cost function on which the dialogue strategy is optimised. Of course, the approach can be used with any function of objectively measurable quantities, and the choice of cost function will usually depend on the nature of the application. We report results for 4 cost functions of the form

$$Cost = NumTurns + FailCost \times NumFailures \tag{1}$$

where *NumTurns* is the average number of turns per transaction, *NumFailures* is the average number of failures per transaction, and *FailCost* is a weighting factor to which we assign the values 20, 10, 5 and 2.

2.4 State space representations

The choice of state space representation is a difficult and critical part of automatic strategy learning. Knowledge of the current state should provide sufficient information not only for selecting an optimal action, but also to enable the learning algorithm to estimate the expected future reward from that state. In addition, it should contain enough history information for the Markov assumption to be reasonable. On the other hand, the size of the state space (which increases exponentially with the number of state variables) should be kept small enough for the problem to remain tractable.

Since these requirements are not easy to satisfy, it is useful to be able to experiment with different state space representations. We used the variables summarised in Table 3 to construct a number of different state spaces. These are listed in Table 4, along with the corresponding state-action space sizes. The latter are equal to the number of Q values that need to be stored after impossible states and state-action combinations have been pruned out.

All of the representations rely on the idea of restricting the state space size by retaining detailed information only for the slot that is currently in focus. Information about other slots is hidden from the reinforcement learning algorithm, while the algorithm for deciding which slot should be in focus at any given moment (based on this hidden information) is simple enough to be handcrafted.

The simplest state space representation (*Small*) does not store any knowledge related to the reliability of the information in the current slot. Such knowledge is added in the other representations, at the cost of enlarging the size of the state space. The *ConfLevel* variable used in *Small Confidence* and *SmallConf + InfoSource* was designed after studying the policies learned for the *Large Confidence* state space, where it was observed that these policies tended to base their actions on a single confidence threshold. In this way it is possible to keep the state space size down without degrading performance.

2.5 Learning algorithm

2.5.1 Description of algorithm

Since we are trying to find a memoryless policy by describing dialogue as an MDP in an environment that is in reality only partially observable (POMDP) [6], there are arguments in favour of using eligibility trace based methods rather than standard Q-learning [5]. These provide a means of combining Q-learning, where state-action values are updated based on the current estimate of the value for the next state, with Monte Carlo learning where

Action	Description
Mixed Initiative	Mixed initiative query for more information (available if current slot is empty).
System Initiative	Query for information on the current slot (available if current slot is empty).
Explicit confirmation	Confirm the contents of the current slot explicitly.
Confirm all	Confirm the contents of all slots.
Implicit confirmation	Confirm the contents of the current slot implicitly while querying for information on the next slot.
Accept	Accept information in the current slot without confirmation at present. Terminate if a complete transaction has been specified.

Table 2: Actions defined for the application

Variable	Possible values
CurrentFocus	Type, Day, Film, Cinema
Status	Empty, Filled
#Empty	0, 1, 2, 3
#Unconfirmed	0, 1, 2, 3
InfoSource	Default,Negated,Elicited,Unelicited
Confidence	Default, 0.2, 0.3, ..., 1.0
ConfLevel	Default, Low, High

Table 3: Variables used in the different state space representations

state-action values are updated based on the reward at the end of the episode. Q-learning has the advantage of allowing the update for each state to be performed immediately, rather than having to wait until the end of the episode. However, because of its dependence on the next state, it may be more vulnerable to violations of the Markov property.

We used Watkins' $Q(\lambda)$ algorithm with replacing eligibility traces [12]. The algorithm is based on estimating the optimal action value function $Q^*(s,a)$, which describes the expected future reward generated by selecting an action a while in state s and following the optimal policy thereafter. Once this function is known, an optimal policy can be followed by simply selecting the action with the largest Q value in each state.

In order to estimate $Q^*(s,a)$, a function $Q(s,a)$ is initialised arbitrarily and the resulting policy interfaced with the simulation system to generate learning dialogues. After each system action, Q(s,a) is updated using the update equation

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha \delta_t e_t(s,a), \qquad (2)$$

where α is a variable step size parameter, $e_t(s,a)$ is the *eligibility* of the state-action pair (s,a) to be updated, and δ_t is an error term:

$$\delta_t = r_t + \max_a Q_t(s_{t+1},a) - Q_t(s_t,a_t), \qquad (3)$$

with r_t being the observed reward and s_{t+1} the successor state after action a_t was taken in state s_t at time t.

The eligibility changes to reflect the strength of the causal link between the current state and a particular state-action pair (s,a) from earlier in the episode. It decays exponentially with time, and is set to zero whenever an exploratory action is taken:

$$e_t(s,a) = \begin{cases} 1 & \text{if } s = s_t \text{ and } a = a_t \\ 0 & \text{if } s = s_t \text{ and } a \neq a_t \\ \lambda e_{t-1}(s,a) & \text{if } s \neq s_t. \end{cases} \qquad (4)$$

Here λ is the eligibility trace decay parameter, which determines the nature of the algorithm. For $\lambda = 0$ it reduces to Q-learning, with action values being updated based only on the immediate successor state. For $\lambda = 1$ the algorithm is a version of Monte Carlo learning, which performs updates based on the eventual accumulated reward. Intermediate values of λ produce algorithms that combine the properties these two approaches.

2.5.2 *Choice of parameters*

The step size was set as $\alpha_k = 1/(p^k)$, where k is the number of times a particular parameter has been updated, and p is a constant between 0.5 and 1.0 (good convergence was obtained with $p = 0.8$). Exploration was done using an epsilon-greedy strategy, with epsilon starting at 1 (random exploration) and decreasing exponentially to a minimum of 0.1 (exploration occurs 10% of the time). These settings satisfy the theoretical requirements to ensure convergence to the optimal policy.

Figure 1 shows how the Q value of the initial dialogue state progresses in a typical training run. After 10^6 episodes the fluctuations are in the order of 0.01, which is sufficient for the purpose of these experiments. The training runs for results presented in the remainder of this paper were performed for a length of 10^6 episodes. To verify the accuracy of these experiments, a number of training runs were also performed for periods of up to 10^7 episodes, with no noticeable change in results.

Policies trained using various values of the eligibility trace decay parameter λ are compared in Figure 2. This illustrates the effect of changing the learning algorithm from Q-learning ($\lambda = 0$) to Monte Carlo learning ($\lambda = 1$) via a set of intermediate algorithms. The graph shows

Name	State variables	Size
Small	Focus, Status, #Empty, #Unconfirmed	209
Information Source	Focus, Status, #Empty, #Unconfirmed, InfoSource	480
Large Confidence	Focus, Status, #Empty, #Unconfirmed, Confidence	1298
Small Confidence	Focus, Status, #Empty, #Unconfirmed, ConfLevel	330
SmallConf + InfoSource	Focus, Status, #Empty, #Unconfirmed, ConfLevel, InfoSource	778

Table 4: State space representations

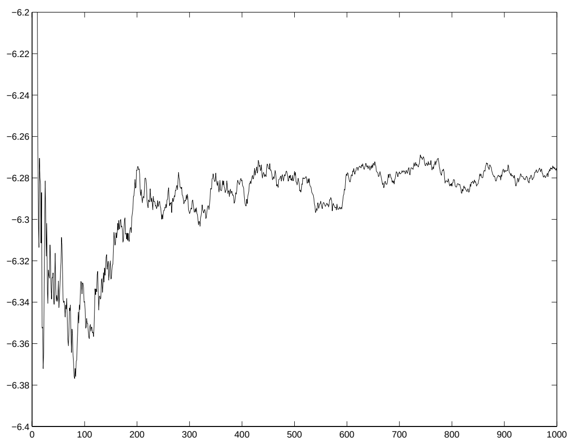

Figure 1: Q value of the initial dialogue state as training progresses

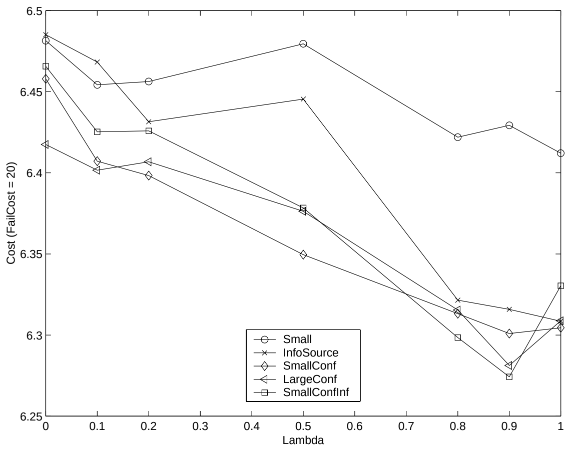

Figure 2: Comparison of policies trained using different eligibility trace decay rates

the performance of policies using each of the state space representations, for the case of $FailCost = 20$. Other cost functions gave similar results.

We observe that different values of λ work well for different state space representations, but overall the best results are obtained for algorithms close to Monte Carlo learning rather than Q-learning, with $\lambda \approx 0.9$. This supports the observation reported in [5] that a value of $\lambda = 0.9$ worked well over a range of problems where memoryless policies were obtained for partially observable MDPs. (Note, however, that the difference in performance for values close to 1 was small and probably not significant. Thus it is not clear whether the use of eligibility traces gives any real improvement over simple Monte Carlo learning.) The remaining experiments presented here use this value for λ.

2.6 Experiments and results

2.6.1 *Baseline system*

The main baseline system available is the original hand designed application. This uses a large number of internal system variables and advanced strategies such as variable confidence thresholds that depend on the number of times specific fields have been prompted for, etc. Since, in order to keep the design problem tractable, we have restricted ourselves to simple state spaces, comparing with this full system is not entirely fair on the more limited automatically designed policies. For this reason we also constructed several handcrafted systems that operate in the *Small* state space, using our defined action set. These policies are simple, not taking much dialogue context into account, but of the type that a human would implement. They are summarised in Table 5.

2.6.2 *Learning experiments*

After training policies for each of the different state space representations and cost functions, the performance of each policy was measured by means of simulations (10^5 episodes per evaluation). The results are shown in Figure 3. Each graph represents the cost, shown on the vertical axis, as measured using a different cost function which is indicated by the label on the left. The horisontal axes represent the policies learned using the different algorithms, which are grouped according to the state space representation used (indicated at the top). The solid lines join policies learned using the same state space representation but different values of *FailCost*.

For each cost function, the policies optimised for that function can be expected to outperform those optimised for a different function. Thus, on the first graph, the policies trained using a *FailCost* of 20 should outperform those using other values. The graph shows that this

Policy	Description
Handcraft 1	Mixed initiative with full confirmation at each step.
Handcraft 2	Mixed initiative with explicit confirmation of each slot but no full confirmation at the end.
Handcraft 3	Mixed initiative with explicit confirmation of each slot and full confirmation at the end.
Handcraft 4	Mixed initiative with confirmation only at the end.
Handcraft 5	System initiative with full confirmation at each step.
Handcraft 6	System initiative with explicit confirmation of each slot but no full confirmation at the end.
Handcraft 7	System initiative with explicit confirmation of each slot and full confirmation at the end.
Handcraft 8	System initiative with confirmation only at the end.

Table 5: Handcrafted policies

is indeed the case, and also that the difference in performance between policies optimised for different cost functions is much greater than those between policies that use different state space representations.

Table 6 compares the results for the optimal policies obtained using the different state space representations, as well as the handcrafted policies in the small state space and the original hand designed system, using a much larger state space. Note that the larger state spaces tend to outperform the smaller ones, but the improvement when extra information is added is small. The automatically designed systems outperform the systems handcrafted on the small state space by large margins. They outperform the original design on some cost functions but are beaten on others, highlighting the inflexibility of the hand design which cannot be optimal for all cost functions.

2.6.3 Comparison of learned policies

The different cost functions represent a range of scenarios where the system is expected to learn dialogue strategies ranging from safe (when the cost of a failure is high, it pays to use confirmation thoroughly) to risky (when failures are less crucial, it is better to streamline the dialogue and use confirmation only when really necessary).

The histogram in Figure 4 shows how the relative frequencies of the different actions in the *Small* state space change as the cost of a transaction failure decreases from 20 to 2. While the exact behaviour of a policy cannot easily be predicted, the large scale trends conform to what one would expect: as the cost of failure decreases, confirmation actions are used less frequently while acceptance actions increase. This illustrates how the learning algorithm adapts to different scenarios in a way that is hard for human designers to emulate.

Figure 5 illustrates how these different policies produce different dialogue costs. The histograms show the cost distributions obtained when the four policies of Figure 4 are evaluated using $FailCost = 5$. The first two policies were optimised for $FailCost = 20$ and $FailCost = 10$ respectively. They have high average dialogue length, which produces a noticeable shift to the right, as well as long tails reflecting the existence of exceptionally long dialogues. These correspond to dialogues with multiple confirmation subdialogues containing recognition errors.

The third policy is the one that has been optimised for this particular cost function. It strikes a balance between minimising the average dialogue length and re-

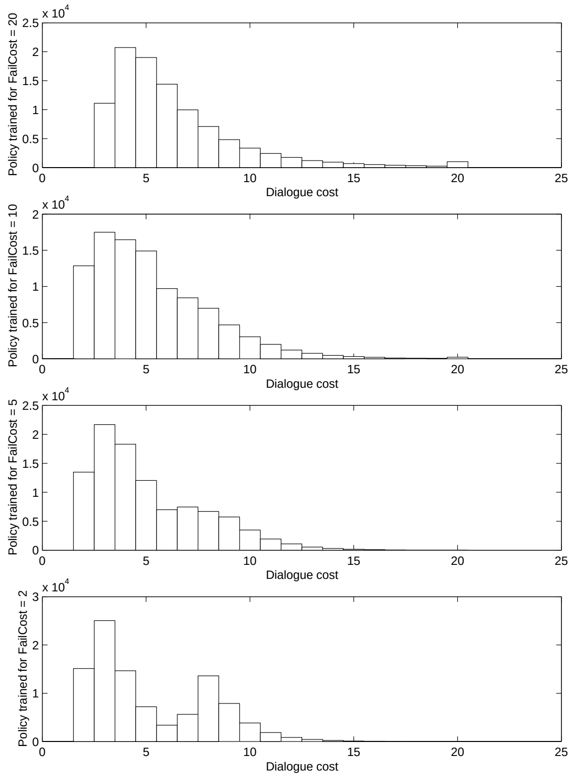

Figure 5: Distribution of dialogue costs for policies optimised for different cost functions: each graph shows a histogram of costs (evaluated using $FailCost = 5$) for policies trained using the cost function indicated on the left

stricting the transaction failure rate, which is visible as a secondary peak emerging to the right. The fourth policy was optimised for $FailCost = 2$ and shortens dialogue length even more, moving the main peak further to the left. However, this comes at the cost of increasing the failure rate such that the secondary peak causes the overall dialogue cost to rise.

3. CONCLUSIONS

The need for dialogue strategies that are adapted to the specific circumstances of the application, such as the characteristics of the user population and the speech

Policy	**Average cost**			
	FailCost = 20	FailCost = 10	FailCost = 5	FailCost = 2
Small	6.43	6.12	5.08	4.27
Information Source	6.32	6.08	5.04	4.24
Large Confidence	6.28	5.84	5.00	4.25
Small Confidence	6.30	5.91	4.98	4.25
SmallConf + InfoSource	6.27	5.86	4.99	4.25
Handcraft 1	9.36	8.79	8.50	8.33
Handcraft 2	11.42	9.99	9.27	8.84
Handcraft 3	9.41	8.78	8.47	8.28
Handcraft 4	9.51	8.84	8.50	8.30
Handcraft 5	8.33	7.75	7.46	7.28
Handcraft 6	10.76	9.27	8.52	8.08
Handcraft 7	8.74	8.04	7.67	7.48
Handcraft 8	8.61	7.77	7.34	7.09
Original hand design	6.82	5.52	4.88	4.49

Table 6: Costs obtained for automatically and manually designed systems

recognition performance, is widely recognised. This paper describes how a detailed user and error simulation tool can be used in an MDP reinforcement learning framework to design such strategies. Since the state space representation can be difficult to design, a great advantage of this approach compared to previous work is the ability to explore different state spaces using a single fixed dialogue corpus, transcribed without reference to system state.

We applied the approach to learn dialogue strategies in a real application, automatically selecting among the many possible approaches to confirmation and mixed initiative issues. A number of state space representations were compared, as well as the effect of using different cost functions. The learned policies outperformed handcrafted policies using the same state space, and gave performance similar to that of the original hand design, which used a much richer state space and action set. They also display flexibility, optimising their behaviour according to the cost function being used.

In a real application it will often not be clear what the best choice of cost function is. Ideally, a cost function should be constructed based on user satisfaction studies done in the specific application domain, using methods such as the PARADISE framework [16]. Where such studies have not been done, the designer may have to choose a cost function in an ad hoc way. This is not as disadvantageous as it may seem, since the behaviour of the learned strategy can then be evaluated qualitatively and the cost function adjusted until the learned strategy displays the desired characteristics. Subsequent user trials can then be used to elicit user satisfaction ratings which can be used to refine the cost function further.

Since the automatically learned strategies yield performance similar to the manually designed system, it would be possible to save several design iterations without degrading performance by using only automatic learning. However, even better performance should be obtainable by combining the two approaches: automatic learning is ideal for optimising low level aspects of system behaviour such as determining appropriate confidence levels, confirmation strategy and management of initiative. Optimal treatment of these aspects depend on the user and error parameters encapsulated in the simulation tool and cannot be designed by hand in a principled way.

Methods such as manual design and automated planning, on the other hand, are useful for high level aspects of system design that are not influenced as directly by the simulation parameters. For instance, the order in which the system should request different pieces of information should not be decided by the reinforcement learner, as it is unlikely that the user model will contain information that is particularly relevant to the question. Instead, the issue should be resolved at a higher level without giving the learning algorithm a choice in the matter.

Such integration of automatic learning methods with more traditional approaches can be done by appropriate selection of the state space and action set that is presented to the reinforcement learner: by restricting the options available to the algorithm, it is allowed to focus on the design aspects for which its models contain useful information, without having to try to distinguish between high level strategies for which it is not suited. At the same time, this approach makes the problem more tractable so that a more detailed state space can be used. Closer investigation into the combination of automatic learning with other methods is an important topic for future research.

Another open research question concerns the refinement of the user model, for instance by using separate models for different subsets of the user population, e.g. a naive and an experienced user group. While the idea of basing the system design on an explicit model of user expertise is not new [13], no data-driven versions of such an approach seem to have been investigated yet. Once such models have been constructed, it would be possible to learn strategies that estimate and adapt to user expertise.

The state space of the user model could also be expanded to include more detailed parameters such as the number of times the user has been asked a specific question. Modelling user behaviour this closely is likely to suffer from data sparsity problems, but if accurate models can be constructed it would expand the range of situations to which the learning system is applicable.

We conclude that the automatic learning approach presented here opens a large scope of possibilities for development and optimisation of adaptive dialogue applications.

4. REFERENCES

[1] D. Goddeau and J. Pineau. Fast reinforcement learning of dialog strategies. *Proc. ICASSP*, 2000.

[2] E. Levin and R. Pieraccini. A stochastic model of computer-human interaction for learning dialogue strategies. *Proc. Eurospeech*, pages 1883–1886, 1997.

[3] E. Levin, R. Pieraccini, and W. Eckert. A stochastic model of human-machine interaction for learning dialogue strategies. *IEEE Transactions on Speech and Audio Processing*, 8(1):11–23, 2000.

[4] D. Litman, M. Kearns, S. Singh, and M. Walker. Automatic optimization of dialogue management. *Proc. COLING*, 2000.

[5] J. Loch and S. Singh. Using eligibility traces to find the best memoryless policy in partially observable Markov decision processes. *The 15th International Conference on Machine Learning,* July 1998.

[6] N. Roy, J. Pineau, and S. Thrun. Spoken dialogue management using probabilistic reasoning. *Proc. Association for Computational Linguistics*, 2000.

[7] K. Scheffler. *Automatic design of human-machine dialogue systems*. PhD thesis, Cambridge University, 2002.

[8] K. Scheffler and S. Young. Corpus-based dialogue simulation for automatic strategy learning and evaluation. *Proc. NAACL Workshop on Adaptation in Dialogue*, 2001.

[9] S. Singh, M. Kearns, D. Litman, and M. Walker. Reinforcement learning for spoken dialogue systems. *Proc. NIPS*, 1999.

[10] S. Singh, M. Kearns, D. Litman, and M. Walker. Empirical evaluation of a reinforcement learning spoken dialogue system. *Proc. AAAI*, 2000.

[11] S. Singh, D. Litman, M. Kearns, and M. Walker. Optimizing dialogue management with reinforcement learning: Experiments with the NJFun system. *Journal of Artificial Intelligence Research*, 16, 2002.

[12] R. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, Massachusetts; London, England, 1998.

[13] G. Veldhuizen van Zandten. User modelling in adaptive dialogue management. *Proc. Eurospeech*, 1999.

[14] M. Walker. An application of reinforcement learning to dialogue strategy selection in a spoken dialogue system for email. *Journal of Artificial Intelligence Research*, 12:387–416, 2000.

[15] M. Walker, J. Fromer, and S. Narayanan. Learning optimal dialogue strategies: a case study of a spoken dialogue agent for email. *Proc. Association for Computational Linguistics*, pages 1345–1352, 1998.

[16] M. A. Walker, D. J. Litman, C. A. Kamm, and A. Abella. PARADISE: A general framework for evaluating spoken dialogue agents. *Proc. Conf. of the Association for Computational Linguistics*, pages 271–280, 1997.

[17] S. Young. Probabilistic methods in spoken dialogue systems. *Philosophical Transactions of the Royal Society (Series A)*, 358:1389–1402, 2000.

[18] B. Zhang, Q. Cai, J. Mao, and B. Guo. Planning and acting under uncertainty: A new model for spoken dialogue systems. *Uncertainty in Artificial Intelligence: Proceedings of the 17th Conference,* pages 572–579, 2001.

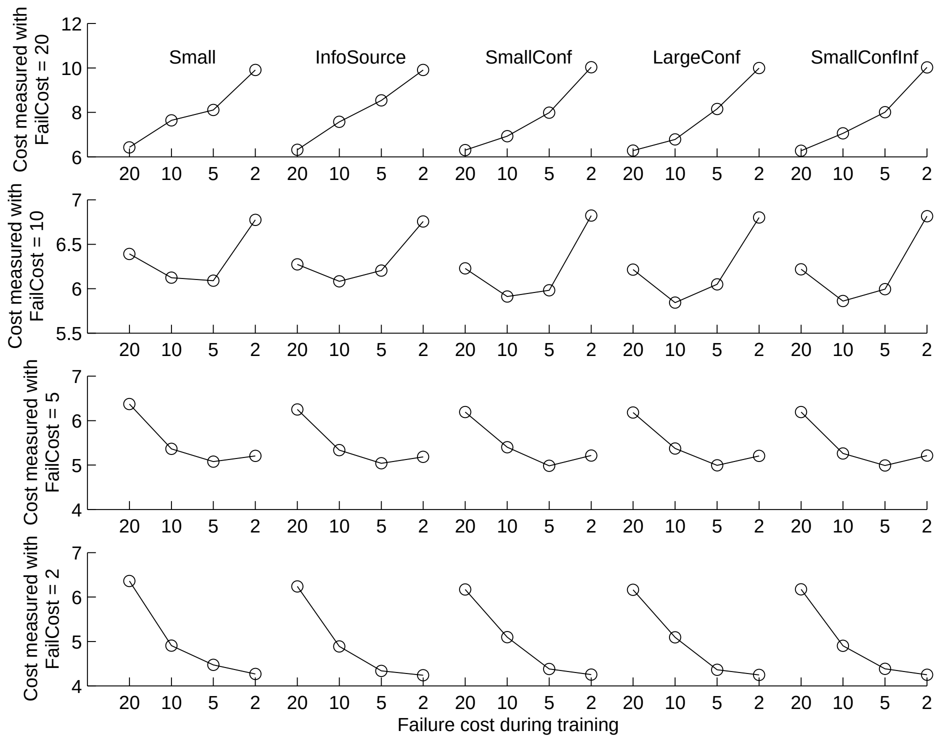

Figure 3: Comparison of policies trained on different cost functions for all state space representations: each graph plots the cost for all the policies, measured using the cost function indicated on the left

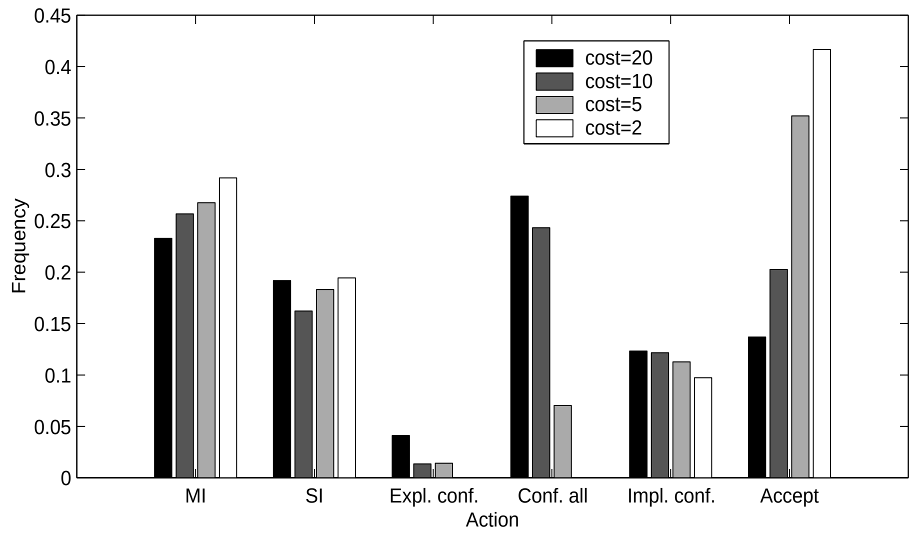

Figure 4: Relative frequencies of actions in policies optimised for different cost functions