CLASSiC

D1.4.2: Summary-based policy optimisation

F. Jurčíček, M. Gašić, B. Thomson, F. Lefèvre, S. Young

Distribution: Public

CLASSiC
Computational Learning in Adaptive Systems for Spoken Conversation
216594 Deliverable 1.4.2

February 2010

Project funded by the European Community
under the Seventh Framework Programme for
Research and Technological Development

The deliverable identification sheet is to be found on the reverse of this page.

Project ref. no.	216594
Project acronym	CLASSiC
Project full title	Computational Learning in Adaptive Systems for Spoken Conversation
Instrument	STREP
Thematic Priority	Cognitive Systems, Interaction, and Robotics
Start date / duration	01 March 2008 / 36 Months

Security	Public
Contractual date of delivery	M24 = February 2010
Actual date of delivery	February 2010
Deliverable number	1.4.2
Deliverable title	D1.4.2: Summary-based policy optimisation
Type	Report
Status & version	Final 1.1
Number of pages	29 (excluding front matter)
Contributing WP	1
WP/Task responsible	WP1, leader UCAM
Other contributors	UCAM: Kai Yu, François Mairesse, Simon Keizer
Author(s)	UCAM: Filip Jurčíček, Milica Gašić, Blaise Thomson, Fabrice Lefèvre, Steve Young
EC Project Officer	Philippe Gelin
Keywords	spoken dialogue management, reinforcement learning, partially observable Markov decision processes

The partners in CLASSiC are:

Heriot-Watt University	HWU
University of Cambridge	UCAM
University of Geneva	GENE
Ecole Superieure d'Electricite	SUPELEC
France Telecom/ Orange Labs	FT
University of Edinburgh HCRC	EDIN

For copies of reports, updates on project activities and other CLASSiC-related information, contact:

The CLASSiC Project Co-ordinator:
Dr. Oliver Lemon
School of Mathematical and Computer Sciences (MACS)
Heriot-Watt University
Edinburgh
EH14 4AS
United Kingdom
O.Lemon@hw.ac.uk
Phone +44 (131) 451 3782 - Fax +44 (0)131 451 3327

Copies of reports and other material can also be accessed via the project's administration homepage, `http://www.classic-project.org`

Contents

Executive summary

This document is a report on summary-based policy optimisation (deliverable 1.4.2), due at month 24 of the CLASSiC project.

It demonstrates summary-based policy optimisation as a viable approach to spoken dialogue management on two dialogue managers: Hidden Information State (HIS) and Belief Update of Dialogue State (BUDS).

An overview of the summary-based policy optimisation and Partially Observable Markov Decision Process (POMDP) Framework is provided: including the implementation of the two POMDP dialogue managers and several improvements to the existing summary-based policy optimisation.

Work related to this deliverable has been published in Young et al. (2009), Thomson and Young (2009), Lefèvre et al. (2009) and Gašić et al. (2009).

1 Introduction

Spoken dialogue systems allow a human user to interact with a machine using voice as the primary communication medium. The structure of a conventional spoken dialogue system (SDS) is shown in Figure 1(a). It contains three major components: speech understanding, speech generation and dialogue management. The speech understanding component typically consists of a speech recogniser and semantic decoder, and its function is to map user utterances into some abstract representation of the user's intended speech act a_u. The speech generation component consists of a natural language generator and a speech synthesiser and it performs the inverse operation of mapping the machine's response a_m back into speech.

The core of the dialogue manager is a data structure which represents the system's view of the world in the form of a machine state s_m. This machine state typically encodes an estimate of three distinct sources of information: the user's input act $\tilde{a}_u$, an estimate of the intended user goal $\tilde{s}_u$[1] and some record of the dialogue history $\tilde{s}_d$[2]. Most conventional dialogue managers rely on hand-crafted deterministic rules for interpreting each (noisy) user dialogue act $\tilde{a}_u$ and updating the state. Based on each new state estimate, a dialogue policy is used to select an appropriate machine response in the form of a dialogue act a_m. This dialogue cycle continues until either the user's goal is satisfied or the dialogue fails.

The designers of such systems have to deal with a number of problems. Since the user's state s_u is unknown and the decoded inputs $\tilde{a}_u$ are prone to errors[3], there is a significant chance that $\tilde{s}_m$ will be incorrect. Hence, the dialogue manager must include quite complex error recovery procedures. Recognition confidence scores can reduce the incidence of misunderstandings but these require thresholds to be set which are themselves notoriously difficult to optimise. Modern recognisers can produce alternative recognition hypotheses but it is not clear in practice how these can be used effectively. Finally, the impact of decisions taken by the dialogue manager do not necessarily have an immediate effect, hence dialogue optimisation requires forward planning and this is extremely difficult in a deterministic framework.

As has been argued previously, taking a statistical approach to spoken dialogue system design provides the opportunity for solving many of the above problems in a flexible and principled way (Young et al., 2009). Early attempts at using a statistical approach modeled the dialogue system as a Markov decision process (MDP) (Levin et al., 1998, 2000; Young, 2000). MDPs provide a good statistical framework since they allow forward planning and hence dialogue policy optimisation through reinforcement learning (Sutton and Barto, 1998). However, MDPs assume that the entire state is observable. Hence, they cannot account for either the uncertainty in the user state $\tilde{s}_u$ and dialogue history $\tilde{s}_d$, or the uncertainty in the decoded user's dialogue act $\tilde{a}_u$.

Figure 1(b) shows an alternative model for the dialogue management component in which the uncertainty in the user's dialogue act and the uncertainty in the machine state are shown explicitly. In this new model, the state estimator maintains a distribution across all states rather than a

[1] Examples of user goals are "finding flight information between London and New York", "finding a Chinese restaurant near the centre of town", "ordering three Pepperoni pizza's", etc.

[2] Since both a_u and s_u are noisy, the record of dialogue history is also noisy, hence the tilde on s_d.

[3] Word error rate (WER) is typically in the 10% to 30% range.

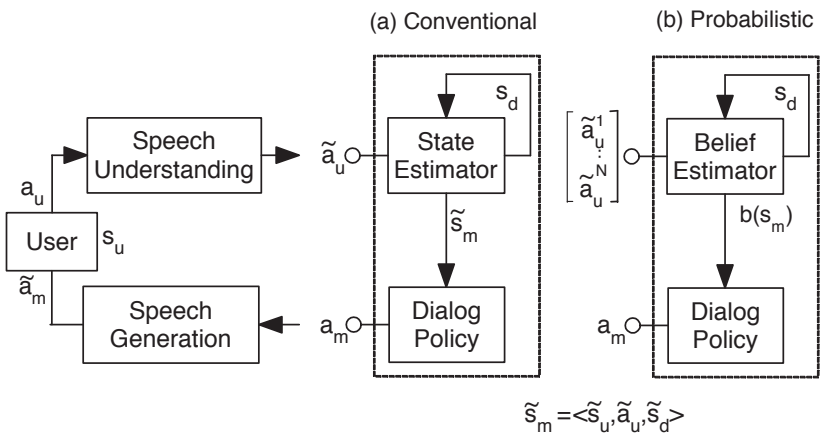

Figure 1: Structure of a spoken dialogue system: a_u and a_m denote user and machine dialogue acts, s_u is the user goal and s_d is the dialogue history. The tilde indicates an estimate. Part (a) shows a conventional dialogue manager which maintains a single state estimate; (b) shows a dialogue manager which maintains a distribution over all states and accepts an N-best list of alternative user inputs.

point-estimate of the most likely state. The dialogue manager therefore tracks all possible dialogue paths rather than just the most likely path. The ensuing dialogue decision is then based on the distribution over all dialogue states rather than just a specific state. This allows competing hypotheses to be considered in determining the machine's next move and simplifies error recovery since the dialogue manager can simply shift its attention to an alternative hypothesis rather than trying to repair the existing one.

If the decoded user input act is regarded as an observation, then the dialogue model shown in Figure 1(b) is a Partially Observable MDP (POMDP) (Kaelbling et al., 1998). The distribution over dialogue states is called the *belief state b* and dialogue policies are based on *b* rather than the estimated state. The key advantage of the POMDP formalism is that it provides a complete and principled framework for modeling the inherent uncertainty in a spoken dialogue system. Thus, it naturally accommodates the implicit uncertainty in the estimate of the user's goal and the explicit uncertainty in the N-best list of decoded user acts. Associated with each dialogue state and machine action is a reward. The choice of reward function is a dialogue design issue, but it will typically provide positive rewards for satisfying the user's goal, and negative rewards for failure and wasting time. As with regular MDPs, dialogue optimisation is equivalent to finding a decision policy which maximises the total reward.

The use of POMDPs for any practical system is, however, far from straightforward. Firstly, in common with MDPs, dialogue states are complex and hence the full state space of a practical SDS would be intractably large. Secondly, since a belief distribution b over a discrete state s of cardinality $n+1$ lies in a real-valued n-dimensional simplex, a POMDP is equivalent to an MDP with a continuous state space $b \in \Re^n$. Thus, a POMDP policy is a mapping from regions in n-dimensional belief space to actions. Not surprisingly these are extremely difficult to construct

and whilst exact solution algorithms do exist, they do not scale to problems with more than a few states/actions.

Several techniques have been developed to tackle this problem most of which are based on the idea of compressing the state space (the master space) into a reduced space (the summary space) so that learning can be tractably performed using approximate algorithms (Williams and Young, 2007b).

The report is structured as follows. Section 2 reviews the theory of POMDP-based dialogue management in general. Section 3 presents two POMDP dialogue systems' implementations. Section 4 deals with policy representation and optimisation using summary state/action space and suggests various improvements to the standard approach. Section 5 evaluates the proposed improvements. The paper ends in section 6 with conclusions.

2 Partially Observable Markov Decision Process

2.1 POMDP Basics

Formally, a POMDP is defined as a tuple $\{S_m, A_m, T, R, O, Z, \lambda, b_0\}$ where S_m is a set of machine states; A_m is a set of actions that the machine may take; T defines a transition probability $P(s'_m|s_m, a_m)$; R defines the expected immediate reward $r(s_m, a_m)$; O is a set of observations; Z defines an observation probability $P(o'|s'_m, a_m)$; λ is a geometric discount factor $0 \leq \lambda \leq 1$; and b_0 is an initial belief state.[4]

A POMDP operates as follows. At each time-step, the machine is in some unobserved state $s_m \in S_m$. Since s_m is not known exactly, a distribution over states is maintained called a belief state such that the probability of being in state s_m given belief state b is $b(s_m)$[5]. Based on the current belief state b, the machine selects an action $a_m \in A_m$, receives a reward $r(s_m, a_m)$, and transitions to a new (unobserved) state s'_m, where s'_m depends only on s_m and a_m. The machine then receives an observation $o' \in O$ which is dependent on s'_m and a_m. Finally, the belief distribution b is updated based on o' and a_m as follows

$$\begin{aligned} b'(s'_m) &= P(s'_m|o', a_m, b) \\ &= \frac{P(o'|s'_m, a_m, b)P(s'_m|a_m, b)}{P(o'|a_m, b)} \\ &= \frac{P(o'|s'_m, a_m)\sum_{s_m \in S_m} P(s'_m|a_m, b, s_m)P(s_m|a_m, b)}{P(o'|a_m, b)} \\ &= k \cdot P(o'|s'_m, a_m) \sum_{s_m \in S_m} P(s'_m|a_m, s_m)b(s_m) \end{aligned} \quad (1)$$

[4] Here and elsewhere, primes are used to denote the state of a variable at time $t+1$ given that the unprimed version is at time t.

[5] In other words, a belief state b is a vector whose component values give the probabilities of being in each machine state.

where $k = 1/P(o'|a_m, b)$ is a normalisation constant (Kaelbling et al., 1998). Maintaining this belief state as the dialogue evolves is called *belief monitoring*.

At each time step t, the machine receives a reward $r(b_t, a_{m,t})$ based on the current belief state b_t and the selected action $a_{m,t}$. The cumulative, infinite horizon, discounted reward is called the *return* and it is given by

$$R = \sum_{t=0}^{\infty} \lambda^t r(b_t, a_{m,t}) = \sum_{t=0}^{\infty} \lambda^t \sum_{s_m \in S_m} b_t(s_m) r(s_m, a_{m,t}). \tag{2}$$

Each action $a_{m,t}$ is determined by a policy $\pi(b_t)$ and building a POMDP system involves finding the policy π^* which maximises the return. The policy $\pi(b_t)$ can have a form of a deterministic function, which maps a belief state to an action, or a distribution from which actions are drawn for every time t.

2.2 Deterministic policies

The optimal deterministic policy is a function of a continuous multi-dimensional variable and hence its representation is not straightforward. However, it can be shown that for finite horizon problems the value function of the optimal policy is piecewise linear and convex in belief space (Sondik, 1971). Hence, it can be represented by a set of *policy vectors* where each vector v_i is associated with an action $a(i) \in A_m$ and $v_i(s)$ equals the expected value of taking action $a(i)$ in state s. Given a complete set of policy vectors, the optimal value function and corresponding policy is

$$V^{\pi^*}(b) = \max_i \{v_i.b\} \tag{3}$$

and

$$\pi^*(b) = a(\operatorname{argmax}_i \{v_i.b\}). \tag{4}$$

This representation is illustrated in Figure 2(a) for the case of $|S_m| = 2$ and a value function requiring just 3 distinct linear segments. The value function itself is the upper heavy line. In this case, b is a 2-D vector such that $b_1 = 1 - b_2$, hence it can be denoted by a single point on the horizontal axis. The linear segments divide belief space into 3 regions and the optimal action to take in each region is the action associated with the uppermost vector in that region. So for example, if $b < x$ in Figure 2a, then action $a(1)$ would be chosen, if $x < b < y$ then action $a(2)$ would be chosen, and so on.

The optimal exact value function can be found by working backwards from the terminal state in a process called *value iteration*. At each iteration t, policy vectors are generated for all possible action/observation pairs and their corresponding values are computed in terms of the policy vectors at step $t-1$. As t increases, the estimated value function converges to the optimal value function from which the optimal policy can be derived. Many spurious policy vectors are generated during this process, and these can be pruned to limit the combinatorial explosion in the total number of vectors (Kaelbling et al., 1998; Littman, 1994). Unfortunately, this pruning is itself computationally expensive and in practice, exact optimisation is not tractable. However, approximate

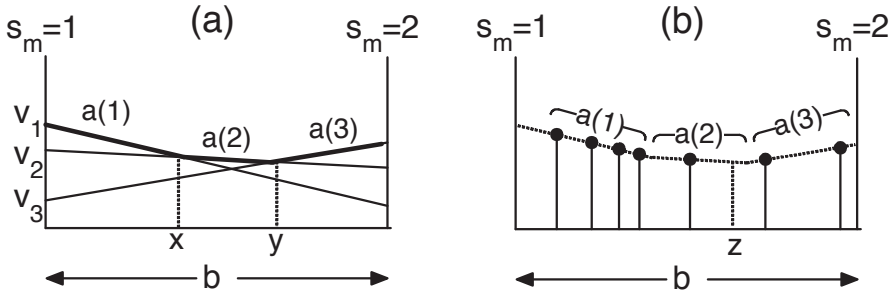

Figure 2: POMDP Value function representation: (a) shows exact an representation; (b) shows a grid-based representation.

solutions can still provide useful policies. The simplest approach is to discretise belief space and then use standard MDP optimisation methods (Sutton and Barto, 1998). Since belief space is potentially very large, grid points are concentrated on those regions which are likely to be visited (Brafman, 1997; Bonet, 2002). This is illustrated in Figure 2(b). Each belief point represents the value function at that point and it will have associated with it the corresponding optimal action to take. When an action is required for an arbitrary belief point b, the nearest belief point is found and its action is used. However, this can lead to errors and hence the distribution of grid points in belief space is very important. For example, in Figure 2(b), if $b = z$ then $a(3)$ would be selected although from Figure 2(a) it can be seen that the optimal action was actually $a(2)$.

Grid-based methods are often criticised because they do not scale well to large state spaces and hence methods which support interpolation between points are often preferred (Pineau et al., 2003). However, the the scaling problem can be avoided by mapping the full belief space into a much reduced summary space where grid-based approximations appear to work reasonably well. Several improvement to this technique are described in section 4.

2.3 Stochastic policies

Instead of using deterministic policies, one can use a stochastic policy $\pi(a_t|\mathbf{b}_t;\boldsymbol{\theta})$, which is a distribution over actions given the belief state depending on the parameters $\boldsymbol{\theta}$.

The task of stochastic policy learning is to find parameters $\boldsymbol{\theta}$ which maximize the long term expected reward which is defined as:

$$J(\boldsymbol{\theta}) = \int p(H;\boldsymbol{\theta})R(H)dH, \tag{5}$$

where $H = \{a_0, o_0, s_0, \ldots, a_T, o_T, s_T\}$ is a trajectory of an episode of the POMDP system, $p(H;\boldsymbol{\theta})$ is a probability of the trajectory H and $R(H)$ is an expected reward for the trajectory H.

The probability $p(H;\boldsymbol{\theta})$ of the trajectory H is defined as:

$$p(H;\boldsymbol{\theta}) = P(s_0)\prod_{t=1}^{T} P(o_t|s_t,a_{t-1})P(s_t|a_{t-1},s_{t-1})\pi(a_{t-1}|b_{t-1};\boldsymbol{\theta}). \tag{6}$$

Learning of $\boldsymbol{\theta}$ can be achieved by a gradient ascent which iteratively adds a multiple of the gradient to the parameters being estimated. The gradient of the long term expected reward is:

$$\nabla J(\boldsymbol{\theta}) = \int p(H;\boldsymbol{\theta}) \nabla \log p(H;\boldsymbol{\theta}) R(H) dH. \tag{7}$$

Generally, the gradient above does not have an analytical solution. Hence, the gradient ascent algorithm must estimate the gradient using a Monte Carlo technique. Assuming that the dialogues are numbered $n = 1,\ldots,N$, and the turns are numbered $0,\ldots,T_n-1$, the equivalent sampled equation is:

$$\nabla J(\boldsymbol{\theta}) = \frac{1}{N} \sum_{n=1}^{N} \sum_{t=0}^{T_n-1} \nabla \log \pi(a_t|b_t;\boldsymbol{\theta}) R(H^n). \tag{8}$$

To improve the estimation of the gradient, a constant baseline, B, can be introduced into the equation above. Williams (1992) showed that the baseline does not introduce any bias into the gradient as $\int \nabla_{\boldsymbol{\theta}} p(H) dH = 0$. Nevertheless, if it is chosen appropriately it lowers the variance of the gradient. As result, the gradient is defined as:

$$\nabla J(\boldsymbol{\theta}) = \frac{1}{N} \sum_{n=1}^{N} \sum_{t=0}^{T_n-1} \nabla \log \pi(a_t|b_t;\boldsymbol{\theta}) (R(H^n) - B). \tag{9}$$

2.4 Summary state/action space

Since exact techniques for POMDP policy learning do not scale well to real-world applications, approximation techniques are necessary to ensure tractability. The concept of the summary state/action space was one of the first ideas developed for this purpose (Williams and Young, 2005).

Beliefs in *master space* are mapped first into summary space and then mapped into a summary action via a dialogue policy. For example, it always makes more sense to confirm the most likely value for a concept rather than any other value. Thus, if the probability of "food=Chinese" is greater than "food=Indian", it would not make sense to attempt to confirm that "food=Indian". The resulting summary action is then mapped back into master space and output to the user. This mapping is necessary because accurate belief monitoring requires that the full propositional content of user goals and dialogue acts be maintained, whereas policy optimisation requires a more compact space.

3 POMDP Spoken dialogue systems

In this section, two approaches to achieving a practical and tractable implementation of a POMDP-based dialogue system using summary state/action space are presented.

First, the Hidden Information State (HIS) dialogue manager (Young et al., 2009) retains a full and rich state representation but only maintains probability estimates over the most likely states.

Conceptually, this approach can be viewed as maintaining a set of dialogue managers executing in parallel where each dialogue manager follows a distinct path. At each dialogue turn, the probability of each dialogue manager representing the true state of the dialogue is computed and the system response is then based on the probability distribution across all dialogue managers.

Second, the Belief Update of Dialogue State (BUDS) dialogue manager (Thomson and Young, 2009) factors the state into a number of simple discrete components. It then becomes feasible to represent probability distributions over each individual factor. The most obvious examples of these are so-called *slot filling* applications where the complete dialogue state is reduced to the state of a small number of slots that require to be filled (Williams and Young, 2007a,b). For more complex applications, the assumption of independence between slots can be relaxed somewhat by using dynamic Bayesian Networks (Thomson et al., 2008b,a). Provided that each slot or network node has only a few dependencies, tractable systems can be built and belief estimates maintained with acceptable accuracy using approximate inference (Bishop, 2006).

3.1 HIS Dialogue manager

The Hidden Information State (HIS) system (Young et al., 2009) uses a full state representation where the state consists of the user goal, the dialogue history and the user action. The user goal is represented as a tree structure – a *partition*, which is built according to a domain ontology. Slots and values are both represented as nodes in the tree. An example of a partition is given in Figure 3. The key idea is that similar user goals are grouped into partitions and a single belief is maintained for each partition.

The grouping of similar user goals is based on the observation that at any point in a dialogue, most of the possible user goal states have identical beliefs simply because there has been no evidence offered by the user to distinguish them. For example, if the system believes that the user might have asked for a Chinese restaurant then it must record `food=Chinese` as a possible goal state, but there is no point in maintaining individual goal states for all of the other possible food types such as `food=Italian`, `food=French`, etc. which have not been mentioned since they are all equally likely.

When querying the data base using the partition, a set of matching entities is produced. The dialogue history consists of the grounding states of nodes in the partition, generated using a finite state machine and the previous user and system action. The combination of a partition, a user action from the last N-best input and the respective set of grounding states forms a *hypothesis*. A distribution over all hypotheses (the *belief state*) is maintained throughout the dialogue. Taking into account that any real-world problem would have a non-trivial ontology, it is clear that this belief space will be extremely large, and must therefore be reduced to a smaller scale summary space.

3.1.1 Summary state/action space

An example of the summary space mapping and action selection process is given in Figure 4. In the top part of the diagram, it is shown how belief b in master space is mapped to a summary point

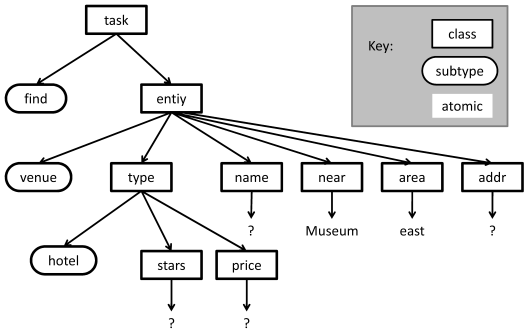

Figure 3: Example of a partition built according to the domain ontology for the goal `find(type=hotel, area=east, near=Museum)`

$\hat{b}$ that contains only five components which are a hybrid of continuous probability values and discrete variables. These are the probability of the 1st and 2nd best dialog hypotheses, a summary status of the top partition (initial, generic, unique , etc.) and a summary of the dialog history of the top hypothesis (initial, supported, offered, etc.) the type of the user act associated with the most likely partition. The distance metric to compare two dialog points strongly encourages discrete components to match exactly by returning a large distance if any discrete component differs, and a smaller Euclidean distance between the continuous components.

Since the summary state space does not encapsulate all the information that master actions can contain, these actions cannot be learnt directly. For that reason, the action space is also reduced to a summary action set. Figure 4 shows how the policy π determines on the basis of summary point $\hat{b}$ which of the ten possible summary actions to take.

The HIS system implements 11 summary actions:

- Bye - the system says good bye and hangs up,
- Confirm - confirms values from the best hypothesis,
- Confirm&Request - implicitly confirms values from the best hypothesis and request the missing values,
- ChooseBetweenTop2Hyp - the system asks the user to select between the two matching entities,
- FindAlternativeHypothesis - the system offers an alternative to the just offered entity,
- Greet - the system says hello,
- Inform - the system informs about some property of an offered entity,
- Offer - the system offers a new entity from the database,

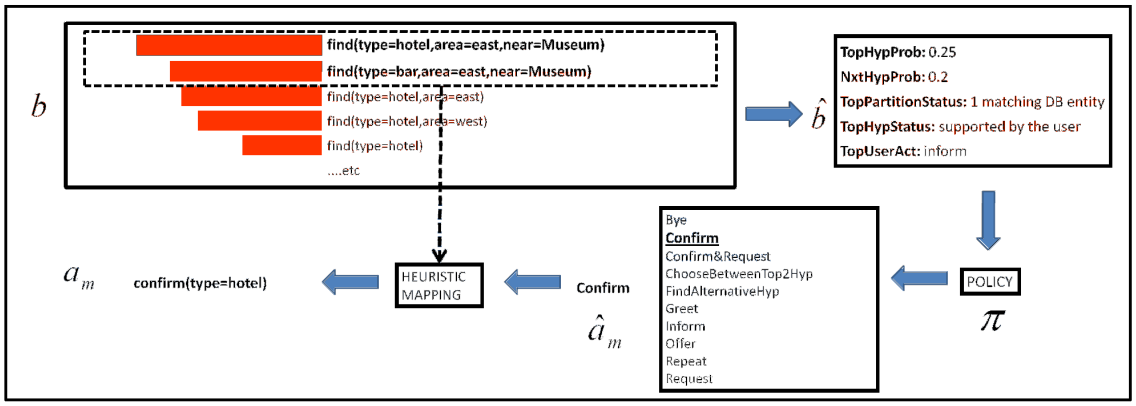

Figure 4: Action selection process in the HIS framework

- Repeat - the system repeats the last system act,
- Request - the system request a missing value from the top hypothesis.

3.1.2 Policy learning and mapping from the summary action space

The optimal policy is obtained using reinforcement learning in interaction with an agenda-based simulated user (Schatzmann et al., 2007). At the end of each dialogue, a reward is given to the system based on the completion of the user goal and the efficiency of the dialogue (20 is given for a successful completion and -1 for each turn). Policy optimisation uses the grid-based Monte Carlo Control algorithm which is described in section 4.1. At each turn the belief is mapped to a summary point from which a summary action can be determined using the policy. The summary action is then mapped back to a master action by adding the relevant information.

The mapping from summary action back to a master action uses heuristics to add the appropriate information from the master space. For example, if the proposed summary action is `Confirm`, then the slot/value pair relating to the node from the partition of the top hypothesis that is not grounded will be confirmed. For instance, if the node relating to the value `hotel` is not grounded the action would be `confirm(type=hotel)`, meaning *You are looking for a hotel, right?*

However, sometimes it is impossible to construct a full master action from the proposed summary action and the information from the master space. For example, it can happen that a summary action such as ”confirm” proposed by policy cannot be mapped back into master space as there might be no information which can be confirmed. As a result, some back-off strategy has to be in place. The default back-off strategy in the HIS system is to use the `Repeat` summary action. In this case, users typically repeat the last dialogue act. For more details on back-off strategies in summary space see section 4.3.

3.2 BUDS Dialogue manager

Similarly to the HIS system, the BUDS's system state is factored into three components: the user goal g, the user action u and the dialogue history h. In addition to it, the BUDS dialogue manager then further factorises according to a set of slots $i \in I$ into sub-goals, sub-user acts, and sub-histories. These are denoted as g_i, u_i, and h_i. In a tourist information system, for example, typical sub-goals might be the type of "food" or the "area". Next some conditional independence assumptions are taken and are modeled as a Bayesian Network (Thomson and Young, 2009). It then becomes feasible to represent probability distributions over each individual slot. Figure 5 shows the resulting network for two time-slices of a two-slot system based on this idea.

In addition to sub-user acts u_i, it is useful to also maintain a node for the overall user act u. To simplify the updates, a one-to-one mapping is defined between the sub acts, u_i and this overall act. Then the sub-acts represent parts of the action that are relevant for a particular sub-goal. As an example, if the user says "I would like a Chinese restaurant", the overall act might be represented as "inform(type=restaurant, food=Chinese)". This will be split so that the sub-act for the *type* concept is "inform(type=restaurant)", while the sub-act for *food* is "inform(food=Chinese)". Acts which cannot be split are associated with every concept. An example of the latter is the user saying "Yes". This will result in an "affirm" act being associated with all slots.

The history node allows the system designer to store sufficient information about the history in order to make coherent decisions. This may be very restricted and in many cases only requires a few values. Indeed, the three states "nothing said", "mentioned" and "grounded" will normally be adequate.

The dependencies of the sub-goals must be limited to enable tractability. A useful method of implementing this is to add a validity node, v_t, for each concept. This node has a deterministic dependency on its parent (or parents) which decides whether the associated sub-goal is relevant to the overall user goal or not. Validity nodes can only take two values - "Applicable" and "Not Applicable". If a node is "Not Applicable" then the associated user sub-goal is forced to also be "Not Applicable". Otherwise the user sub-goal will depend on its previous value with some probability of change. Figure 5 shows the resulting network for a system with two concepts: type (of venue) and food. Note that the user goal and user act are assumed to be conditionally independent of the history.

Tractability is a major concern whenever a statistical model of uncertainty is used since statistical models require that a probability be associated with every possible state of the environment. Unless significant assumptions and approximations are taken, this is intractable for any real-world system. The BUDS system's inference algorithm is based on the loopy-belief propagation (LBP) algorithm (Frey and MacKay, 1998). LBP enables tractability by exploiting the conditional independence of concepts in the network.

By grouping together slot values, significant improvements over the original algorithm are obtained. The grouped LPB maintains marginal belief estimates only for slot values which were mentioned by a user. Those slot values which were not mentioned are assumed to belong to one group with an uniform probability distribution within this group. Then, the belief update involves only recomputation of probabilities of the mentioned slot values and one probability of

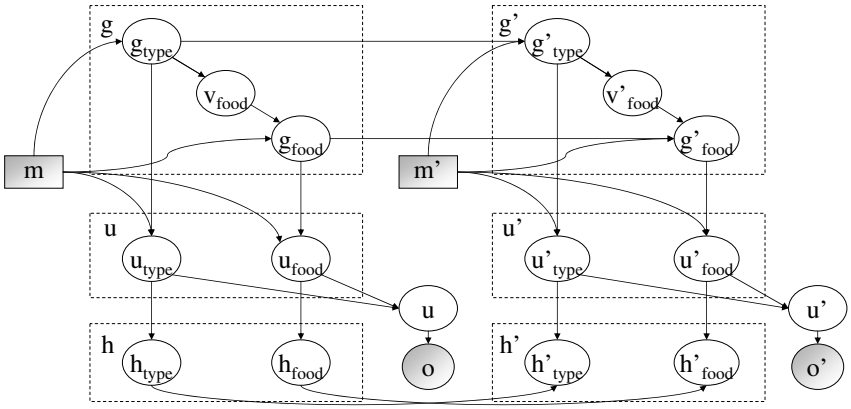

Figure 5: An example factorisation for the Bayesian network representing part of a tourist information dialogue system.

the grouped values. The grouped LPB usually helps in situations when the number of potential slot values is in hundreds and a user mention only several of them. Further efficiency gains are obtained by assuming limited changes in the user goals (Thomson and Young, 2009). The result is then a framework which is able to cope with dependencies as well as very large numbers of concept values. As such, it becomes a feasible basis for building real-world systems which can maintain a fully Bayesian belief state in real-time.

3.2.1 Summary action space

In comparison with the HIS system, the BUDS system does not have an explicit summary state space as it uses approximation techniques for the stochastic policy. However, it extracts a set of features from the belief state which helps to decide which summary action to take. In addition, BUDS also differ in the set of summary action. As it models marginal probabilities for each slot value, it enables the BUDS system to implement component based policy. In such policy, there are slot summary actions and a set of global summary actions. There are 3 slot summary actions for each slot in the BUDS system:

- Request - the system requests a slot value,
- Select - the system selects between the two most likely values,
- Confirm - the system explicitly confirms the most likely value.

The set of global actions includes:

- Bye - the system says good bye,
- ReqMore - the system informs about non-accepted slots for the informed entity,

- Repeat - the system repeats the last system's act,
- InformAlternatives - the system informs about alternatives to already offered entities,
- InformExact - the system informs about slots which were accepted for the matching entity,
- InformRequested - the system informs about the requested slots or confirm them.

For each summary action, a, a *basis function* is defined, $\boldsymbol{\phi}_a$, which is a vector function of the belief state, b. The basis function gives a set of features from the belief state that are important for decision making.
The BUDS system uses a softmax function to define a stochastic policy:

$$\pi(a|b,\boldsymbol{\theta}) = \frac{e^{\boldsymbol{\theta}\cdot\boldsymbol{\phi}_a(b)}}{\sum_{a'} e^{\boldsymbol{\theta}\cdot\boldsymbol{\phi}_{a'}(b)}}, \tag{10}$$

where the parameters, $\boldsymbol{\theta}$, and the basis function, $\boldsymbol{\phi}_a$, determine the likelihood of taking an action. The exact features used in the system depends on the application. Since the belief state is a vector of the marginal distributions for each slot, $i \in I$, in the network [6], one can then separate the features into components, denoted $\boldsymbol{\phi}_{a,i}(b)$. Each slot-level function must encode a set of features that depends on the machine action and the slot. Extra information is encoded in an overall basis function, $\boldsymbol{\phi}_{a,*}(b)$. The full basis function can then be as follows:

$$\boldsymbol{\phi}_a(b)^\top = [\boldsymbol{\phi}_{a,1}(b)^\top, \ldots, \boldsymbol{\phi}_{a,K}(b)^\top, \boldsymbol{\phi}_{a,*}(b)^\top]. \tag{11}$$

3.2.2 Grid-based slot level features

A straightforward way of defining $\boldsymbol{\phi}_{a,i}(b)$ is to use grid-based approximation. For each slot, the probabilities of the two most likely values are formed into a tuple, $(p1, p2)$. For example, a collection of six grid points is defined: $\{(1,0),(0.8,0),(0.8,0.2),(0.6,0.0),(0.6,0.2),(0.6,0.4)\}$, numbered $1,\ldots,6$. Grid-based features, denoted $y_{i,l}$, are then defined for the values $l = 0,\ldots,6$:

$$\begin{aligned} y_{i,l} &= \begin{cases} 1 & \text{if } p1 \geq 0.4 \text{ and } l \text{ is the index of the closest point in the set} \\ 0 & \text{otherwise} \end{cases} \\ y_{i,0} &= \begin{cases} 1 & \text{if } p1 < 0.4 \\ 0 & \text{otherwise} \end{cases} \end{aligned} \tag{12}$$

Consequently, the slot level function is composed as follows: $\boldsymbol{\phi}_{a,i}(b) = \{y_{i,0},\ldots,y_{i,6}\}$ if the machine action m is a slot summary action operating on the slot i otherwise $\boldsymbol{\phi}_{a,i}(b) = \{0,\ldots,0\}$.

[6] Given variables indexed by $i = 1,\ldots,N_i$, marginals denoted b_i, the overall belief state would be a vector $b = (b_1,\ldots,b_{N_i})^T$

The overall basis function, $\phi_{a,*}(b)$ contains number of matching venues, n_{venues}. This number can be again discretised by a binary vector:

$$\begin{aligned} y_{*,0} &= \begin{cases} 1 & \text{if } n_{venues} = 0 \\ 0 & \text{otherwise} \end{cases} \\ y_{*,1} &= \begin{cases} 1 & \text{if } n_{venues} = 1 \\ 0 & \text{otherwise} \end{cases} \\ y_{*,2} &= \begin{cases} 1 & \text{if } n_{venues} > 1 and n_{venues} < 4 \\ 0 & \text{otherwise} \end{cases} \\ y_{*,3} &= \begin{cases} 1 & \text{if } n_{venues} \geq 4 \\ 0 & \text{otherwise} \end{cases} \end{aligned} \tag{13}$$

Then, the function $\phi_{a,*}(b)$ is composed as follows: $\phi_{a,i}(b) = \{y_{*,0}, \ldots, y_{*,3}\}$.

3.2.3 Entropy-based slot level features

Another way to define the slot level features is based on the entropy, $entropy(i)$, of a marginal probability distribution of a slot i. For example, if the slot entropy is high then it is natural to ask a user to provide a value of that slot as the entropy mean that there is high uncertainty among the slot values. If the slot entropy is low them the most likely vale is likely to be correct and it can be accepted. However, the entropy cannot be the only feature used as the entropy value does not uniquely characterise the slot distribution. So, it is accompanied by the probability of the first and second most likely value in a slot, $p_1(i), p_2(i)$. As a result, the slot level function is composed as follows: $\phi_{a,i}(b) = \{entropy(i), p_1(i), p_2(i)\}$ if the machine action m is a slot summary action operating on the slot i otherwise $\phi_{a,i}(b) = \{0, \ldots, 0\}$.
The overall basis function $\phi_{a,*}(b)$ can be the same as in the previous section. The advantage of the entropy-based basis function that that it smaller set of features and while retaining the performance of the grid-based features. For more details, see section 4.4.

3.2.4 Policy learning and mapping from the summary action space

Similarly to the HIS system, the stochastic policy used in the BUDS system can be optimised using reinforcement learning in interaction with an agenda-based simulated user and a reward based on task completion. Policy optimisation uses the Natural Actor Critic algorithm (Peters et al., 2005) which is described in section 4.4. At each turn features are extracted from the the belief state and then a random summary action is sampled from the stochastic policy. Then, the summary action is mapped back to a master action by adding the relevant information. For example, in the BUDS system a ”confirm food” action would be a summary act. Given a belief state, the corresponding master space action would be to confirm the most likely food type.

4 Policy Optimisation Methods

The essential role of the dialogue manager is to decide what action to take at each turn. This depends on the policy, which maps belief states to actions. This mapping can either be a deterministic function which always assigns one action to the belief state (see section 2.2) or the actions can be drawn from a stochastic policy - a distribution over actions given the belief state (see section 2.3).

In commercial systems, the policy is usually hand-crafted. At each point in the call flow, the dialogue designer chooses which action the system should take. Hand-crafted policies are also common for systems which use a statistical approach to uncertainty. After each turn the system updates its belief using probability theory, possibly with the techniques mentioned in 3.1 or 3.2. However, hand-crafting policies is time consuming and can result in sub-optimal decisions. As a result, various researchers have attempted to overcome these problems by learning policies automatically.

This section presents several approaches to automatic policy learning developed for the HIS and BUDS systems, which includes use of summary state/action space.

4.1 Grid-based Monte Carlo Policy Optimization

In a POMDP, the optimal exact value function can be found iteratively from the terminal state in a process called *value iteration*. At each iteration t, policy vectors are generated for all possible action/observation pairs and their corresponding values are computed in terms of the policy vectors at step $t-1$. However, exact optimisation is not tractable in practice, but approximate solutions can still provide useful policies. Representing a POMDP policy by a grid of representative belief points yields an MDP optimisation problem for which many tractable solutions exist, such as the Monte Carlo Control algorithm (Sutton and Barto, 1998) used here.

In the current HIS system (Young et al., 2009), each summary belief point is a vector consisting of the probabilities of the top two hypotheses in master space, two discrete status variables summarising the state of the top hypothesis and its associated partition, and the type of the last user act.

In order to use such a policy, a simple distance metric in belief space is used to find the closest grid point to a given arbitrary belief state:

$$\begin{aligned} |\hat{b}_i - \hat{b}_j| &= \sum_{d=1}^{2} \alpha_d \cdot \sqrt{(\hat{b}_i(d) - \hat{b}_j(d))^2} \\ &+ \sum_{d=3}^{5} \alpha_d \cdot (1 - \delta(\hat{b}_i(d), \hat{b}_j(d))) \end{aligned} \tag{14}$$

where the α's are weights, d ranges over the 2 continuous and 3 discrete components of $\hat{b}$ and $\delta(x,y)$ is 1 iff $x = y$ and 0 otherwise.

Associated with each belief point is a function $Q(\hat{b}, \hat{a}_m)$ which records the expected reward of taking summary action $\hat{a}_m$ when in belief state $\hat{b}$. Q is estimated by repeatedly executing dia-

logues [7] and recording the sequence of belief point-action pairs $\langle \hat{b}_t, \hat{a}_{m,t} \rangle$. At the end of each dialogue, each $Q(\hat{b}_t, \hat{a}_{m,t})$ estimate is updated with the actual discounted reward. Dialogues are conducted using the current policy π but to allow exploration of unvisited regions of the state-action space, a random action is selected with probability ε.

Once the Q values have been estimated, the policy is found by setting

$$\pi(\hat{b}) = \arg\max_{\hat{a}_m} Q(\hat{b}, \hat{a}_m), \qquad \forall \hat{b} \in \mathcal{B} \tag{15}$$

Belief points are generated on demand during the policy optimisation process. Starting from a single belief point, every time a belief point is encountered which is sufficiently far from any existing point in the policy grid, it is added to the grid as a new point. The inventory of grid points is thus growing over time until a predefined maximum number of stored belief vectors is reached.

The main issue in grid-based learning is how to generate grid-points efficiently. The approach used here is to start with a single grid point and then add new points as required during the exploration phase of training. The learning starts by arbitrarily assigning values to the Q-values for each action a associated with the initial grid point $\hat{b}_0$[8]. In addition to the Q values, a counter $N(\hat{b}, a)$ is associated with each grid point and initialised to zero. This counter records the number of times that each action is taken in that grid point. Each learning episode is conducted ε-greedily *i.e.,* using the current best policy ($\pi(\hat{b}) = argmax_a Q(\hat{b}, a)$) except that with probability ε a random action is taken instead of the action proposed by the policy. In the standard algorithm, the policy consists of one action per grid point – this is the action that has the highest Q-value (Sutton and Barto, 1998). For the N-best action selection, the algorithm is modified so that the policy consists of a list of actions per grid point ordered by their Q-values. In a similar way, when exploring instead of generating one random action, a random ordering of actions is generated. In all cases, the reward obtained for each turn is assigned to the Q-value for the action that was actually taken. Every time a new belief state is visited, it is mapped to a summary state and then to the nearest grid point. If there is no nearby grid point a new one is created and added to the set of grid points. Thus, during training grid points are created with their respective lists of Q- and N-values.

4.2 k-nn Monte Carlo Policy Optimization

k-nn Monte Carlo Policy Optimization (Lefèvre et al., 2009) extends the previous algorithm so as to reduce training over-fitting and to improve robustness to noise in the user input. Similarly to the Grid-based Monte Carlo Control algorithm, this technique uses a database of belief vector prototypes to choose the optimal system action. However, a locally weighted *k*-nearest neighbor scheme is introduced to smooth the decision process by interpolating the value function, resulting in higher user simulation performance. This method thus lies between a strictly grid-based and a point-based value iteration approach as it interpolates the value function around the queried belief

[7] A simulator is often used for this to allow efficient training over a large number of dialogues.

[8] The hat on variables denotes elements the summary space.

point. It thus reduces the policy's dependency on the belief grid point selection and increases robustness to input noise.

The algorithm maintains a set of sample vectors $\hat{b}$ along with their Q value vector $Q(\hat{b},a)$. When a new belief state $\hat{b}'$ is encountered, its Q values are obtained by looking up its k-nearest neighbours in the database, then averaging their Q-values. To obtain good estimates for the value function interpolation, local weights are used based on the belief point distance. A Kullback-Leibler (KL) divergence (relative entropy) could be used as a distance function between the belief points. However, while the KL-divergence between two continuous distributions is well defined, this is not the case for sample sets. In accordance with the locally weighted learning theory (Moore et al., 1997), a simple weighting scheme based on a nearly Euclidean distance (eq. 14) is used to interpolate the policy over a set of points:

$$\pi_{\text{knn}}(\hat{b}) = \arg\max_{\hat{a}_m} \sum_{\{\hat{b}_k\}_{\text{knn}}} Q(\hat{b}_k, \hat{a}_m) \times \Phi(\hat{b}_k, \hat{b}).$$

There are various ways how to set the weighting coefficients but one that has worked well in the framework presented here is the following kernel function:

$$\Phi(\hat{b}_1, \hat{b}_2) = e^{-|\hat{b}_1 - \hat{b}_2|^2}.$$

The policy learning procedure k-nn Monte Carlo Control algorithm is similar to the original Monte Carlo Control algorithm presented in the previous section. Only the the part when actions are proposed has to utilise Q-values from the neighbourhood of a belief point.

4.3 N-best Back-off action selection

This section deals with the issue of invalid state-action pairs in the Partially Observable Markov Decision Process (POMDP) framework (Gašić et al., 2009). In particular, when modeling dialogue as a POMDP, both the state and the action space must be reduced to smaller scale summary spaces in order to make learning tractable. However, since not all actions are valid in all states, the action proposed by the policy in summary space sometimes leads to an invalid action when mapped back to master space. Some form of back-off scheme must then be used to generate an alternative action.

It is a general property of Markov decision processes that not all actions are valid in every state. In a discrete observable MDP, this problem can be resolved simply by defining for each state a subset of actions that are possible. It is then straightforward during training and operation to ensure that the policy for any state only considers actions which are valid for that state.

In contrast, a POMDP belief state represents a probability over all environment states and since in principle any environment state is possible in any belief state, the policy must include the possibility of taking any action in any belief state. For example, in a dialogue system given some prior information on user preferences, there is nothing in principle to stop the system saying as its very first action: "Please confirm that you want a Chinese restaurant?", even though the user hasn't yet said anything. This problem is exacerbated in a system which compresses actions into

simple strategic decisions such as "ask", "confirm", etc. In this case, the fact that an action such as "confirm" in summary space is invalid, cannot be determined until it is mapped back into master space where it is discovered that there is in fact no information which can be confirmed.

To deal with this problem, the optimal back-off action selection should utilise Q-values. The Q-value $Q(b,a)$ is the expected reward from taking action a in belief state b and following the policy thereafter. It can be calculated as $Q(b,a) = \sum_{s \in S} Q(s,a)b(s)$, where $Q(s,a)$ is the Q-value of taking action a in state s. For each action a, the set of states in which that action can be taken is known. Therefore, the learning algorithm can estimate $Q(b,a)$ by summing over only the $Q(s,a)$ values for which action a is valid in s. Ordering $Q(b,a)$ in a list for each belief b provides a sequence of possible back-off actions which can be searched until a valid action is found.

A major problem with POMDPs is the intractability of exact learning algorithms and hence the need for approximate solutions. The Monte Carlo Control algorithm used in the two previous sections use grid-based discretisation of the belief space which further exacerbates the invalid action problem since the effect of the discretisation is to represent all of the belief points in a neighbourhood by a single representative grid point. Thus, even if there are no belief points within the neighbourhood with invalid actions, the merging into a single grid point will generate the union of all neighbouring actions and this union might have invalid actions. One possibility to avoid this is to increase the information transferred from master to summary space with the subset of actions that are valid in that summary state. This possibility is straightforward to implement and it is explored further in section 5.3.

The grid-based the Monte Carlo Control algorithm from the previous sections serves as an initial algorithm. The basic idea is simple. The POMDP summary space is represented by a number of discrete grid points. The system interacts with a user and Q-values are estimated for each grid point and each action. The action with the highest Q-value at each grid point then forms the policy. Extending this algorithm to include N-best action selection is then simply a matter of storing not just the highest Q-value at each grid point but a rank-ordered list (Gašić et al., 2009).

4.4 Natural Actor Critic algorithm

The BUDS system described in section 3.2 utilises the Natural Actor Critic (NAC) algorithm (Peters et al., 2005) to perform policy optimisation over a factorised state space. The NAC algorithm has several advantages over the previously proposed approaches. Most importantly, the use of a factorised policy enables the system to learn to differentiate between both slot-level and higher-level actions simultaneously. Other advantages of the NAC algorithm include its ability to do online policy adaptation and its relatively strong convergence guarantees.

In section 2.3, the equation 9 defines gradient of the expected long term reward. Note that the gradient still uses the expected reward $R(H^n)$ of the trajectory H^n. As the expected reward $R(H^n)$ is not usually easily available, it has to be approximated. For example, Williams (1992) use a reward which is observed at the end of an episode (a dialogue in our case). However, it can be shown that this leads to a biased gradient.

Another approach is to approximate $R(H^n)$ by a function which is compatible with the policy without affecting unbiasedness of the gradient estimate (Konda and Tsitsiklis, 2000). A compat-

ible function approximation of $R(H^n)$ parametrized by a vector w is as follows:

$$R(H^n) \approx \sum_{t=0}^{T_n-1} \nabla_{\boldsymbol{\theta}} \log \pi(a_t|b_t)^T \cdot w + J \tag{16}$$

To compute the parameters w, the expected reward $R(H^n)$ can be replaced by an observed reward r^n at the end of the dialogue $n \in \{1,\ldots,N\}$. Then, the approximation task results into a least square problem:

$$r^n = \sum_{t=0}^{T_n-1} \nabla_{\boldsymbol{\theta}} \log \pi(a_t|b_t)^T \cdot w + J \tag{17}$$

Once the approximation of the expected reward is defined, it can be used in the a gradient of the expected long term reward (equation 9)

$$\nabla J(\boldsymbol{\theta}) \approx \frac{1}{N} \sum_{n=1}^{N} \sum_{t=0}^{T_n-1} \nabla \log \pi(a_t|b_t;\boldsymbol{\theta}) \nabla \log \pi(a_t|b_t;\boldsymbol{\theta})^T \cdot w \tag{18}$$

In the equation above, it was conveniently assumed that the baseline B is equal to J. In fact, it can be shown that J minimize the variance of the gradient as it is expected reward over all trajectories H.

Traditional gradient ascent iteratively adds a multiple of the gradient to the parameters being estimated. This is not necessarily the most effective gradient as it does not always point in the steepest direction. In optimising an arbitrary loss function, $J(\boldsymbol{\theta})$, Amari (1998) shows that for a general Riemann space the direction of steepest ascent is in fact $G_{\boldsymbol{\theta}}^{-1} \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$, where $G_{\boldsymbol{\theta}}$ is a metric tensor. The optimal metric tensor to use in a statistical model is typically the Fisher Information Matrix (Amari, 1998).

Interestingly, Peters et al. (2005) noticed and proved that

$$\frac{1}{N} \sum_{n=1}^{N} \sum_{t=0}^{T_n-1} \nabla \log \pi(a_t|b_t;\boldsymbol{\theta}) \nabla \log \pi(a_t|b_t;\boldsymbol{\theta})^T, \tag{19}$$

taken from the equation 18, is in fact an estimate of the Fisher Information matrix. Consequently, the gradient (equation 18) can be written as follows:

$$\nabla J(\boldsymbol{\theta}) \approx G_{\boldsymbol{\theta}} w. \tag{20}$$

As a result, the natural gradient of the expected long term reward is:

$$\widetilde{\nabla}_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = G_{\boldsymbol{\theta}}^{-1} \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \approx G_{\boldsymbol{\theta}}^{-1} G_{\boldsymbol{\theta}} w = w \tag{21}$$

To conclude, the NAC algorithm solves the least square problem given in the equation 17 to obtain a natural gradient of the expected long term reward $J(\boldsymbol{\theta})$ (see the equation 5) for every update of the parameters $\boldsymbol{\theta}$.

5 Evaluation

The evaluation of proposed algorithms was divided into two groups. The first group was comprised of three experiments evaluating the proposed improvements to the HIS system: the baseline grid-based Monte Carlo algorithm (section 4.1), the *k*-nn Monte Carlo algorithm (section 4.2), and N-best Back-off action selection algorithm (section 4.3). The second group of experiments compared the grid-based slot level features (section 3.2.2) and entropy-based slot level features (section 3.2.3) in the BUDS system using a component based policy.

All the algorithms are evaluated against an agenda based user simulator (Schatzmann et al., 2007) on the Town-Info domain which provides tourist information for an imaginary town. However, no specific domain knowledge is incorporated in the dialogue managers that would not be applicable for any query-driven dialogue.

The reward function used in all cases awards 20 less the number of dialogue turns for a successful dialogue and 0 less the number of turns for an unsuccessful one. A dialogue is considered successful if a suitable venue is offered and all further pieces of information are given. In the case where no venue matches the constraints, the dialogue is deemed successful if the system tells the user that no venue matches and a suitable alternative is offered.

5.1 Grid-based Monte Carlo Policy Optimization

The training schedule adopted by the grid-based Monte Carlo Policy optimization algorithm is comparable to the one presented in Young et al. (2009). The system was trained and tested using a user simulator which incorporates an error model to allow a range of noise levels to be simulated. Training starts in a noise free environment using a small number of grid points and it continues until the performance of the policy asymptotes. The resulting policy is then taken as an initial policy for the next stage in which the noise level is increased, the set of grid points is expanded and the number of iterations is increased. Approximately $1,000,000$ dialogues were used for training and the total number of grid points was 400.

The system implements the fixed back-off strategy which use a single global default summary action as the back-off action. Every time the summary action suggested by a policy cannot be mapped back to the master space, the summary action `Repeat` action is used. In this case the system asks the user to repeat the last input. It is typical that users either repeat the last dialogue act or hang up if the system has already asked this several times. By asking the user to repeat the last act, the system can potentially obtain a better estimation of the current user state, but it can also waste time leading to a lower reward.

The evaluation of the system on the user simulator performing 5000 dialogues at each of 11 error rates is shown in Figure 6 under the name ”Fixed back-off”.

5.2 k-nn Monte Carlo Policy Optimization

The training used the same setup as in the previous section 5.1. The results obtained by the policy trained by *k*-nn Monte Carlo algorithm are in Figure 7. Multiple k from the set $k\{1,3,5,7\}$ were

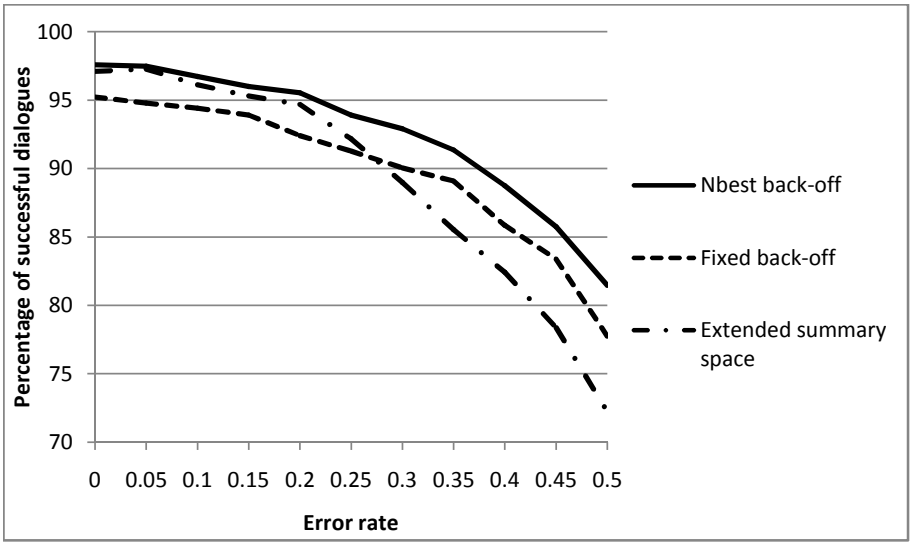

Figure 6: Comparison of the percentage of the successfully completed simulated dialogues between the fixed back-off strategy, the strategy with extended summary space and the N-best back-off strategy on different error rates.

evaluated.

The results demonstrate that the *k*-nn policies outperform the baseline 1-nn policy, especially on high noise levels. However, increasing *k* above 3 does not improve performances. This is likely to be due to the small size of the summary space as well as the use of discrete dimensions.

5.3 N-best Backoff action selection

This section presents results for the proposed N-best back-off action selection method and compares it with a simple fixed back-off baseline (section 5.1) and an alternative based on extending the summary space with features designed to minimise the occurrence of invalid actions.

5.3.1 Extension of the Summary Space

The summary space can be extended with explicit information about which action can be taken so that the occurrence of invalid actions is minimsed. In this approach, each summary grid point is augmented with a binary flag to indicate whether or not each summary action is valid for that grid point. Since two belief points that have different subsets of plausible actions cannot be mapped to the same grid point and since there are 11 possible summary actions, this can potentially increase the summary space by a factor of 2^{11}. In practice, however, due to the nature of the problem some actions are always possible and there are also dependencies between them, so in total the extended summary space resulted in a policy with 2000 grid points. The training scheme used for this system was similar to the training from the section 5.1, with the difference that training had ten times more dialogues to compensate for the increased number of grid points.

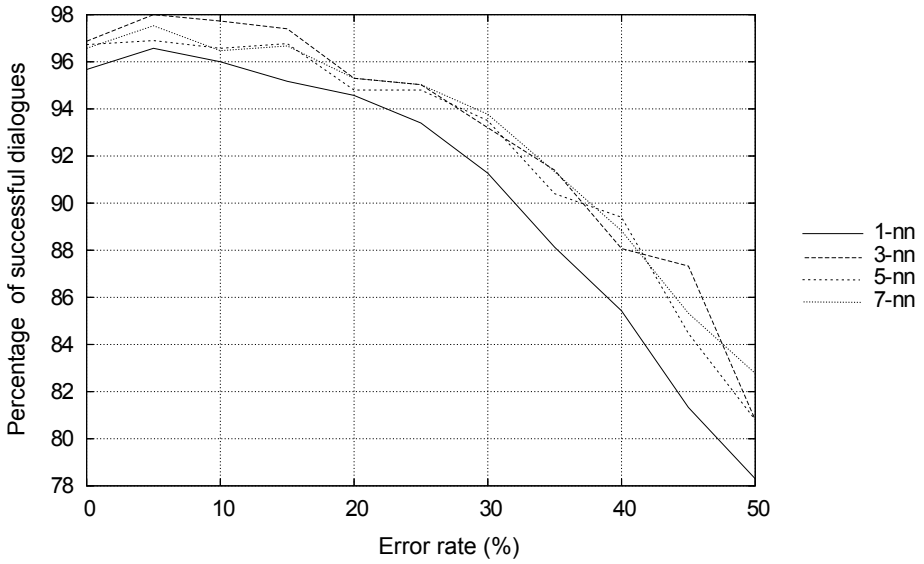

Figure 7: Comparison of the percentage of successful simulated dialogues between the *k*-nn strategies on different error rates.

The performance of this system is shown in Figure 6 under the name ”Extended summary space”.

5.3.2 N-best Back-off

Q values from the Monte Carlo Control algorithm produce a list of actions ranked by the policy preference associated with each summary state (see section 4.3). Utilising this N-best list for back-off action selection, the policy was trained using the same training scheme as in the fixed back-off strategy. The performance results for this system are given in Figure 6 under the name ”N-best back-off”.

As shown, in Figure 6, the N-best back-off outperforms both the fixed back-off and the extended summary space strategies across all error rates, where the extension of the summary space improved the performance on low error rates. However, due to increased fragmentation of summary space, performance degraded rapidly in noise and increasing the number of training dialogues by a factor of 10 was not able to compensate for this. Hence, it appears that extending summary space with features whose only purpose is to avoid invalid actions degrades the policy overall and results in a system with poor robustness to noise.

It is also interesting to note that the frequency with which the top proposed action is not taken differs in the N-best and fixed back-off strategies. The policy obtained using the N-Best back-off strategy backs off more often (see Figure 8). This suggests that this policy has more liberty when choosing the action, since there is a whole list of back-off actions to try if the top action fails. Not only does the N-best strategy back off more, but the percentage of backing-off increases more dramatically than with the fixed back-off strategy. This may be ascribed to the difficulty of correctly determining which actions are valid in noisy states. Therefore, the N-best strategy

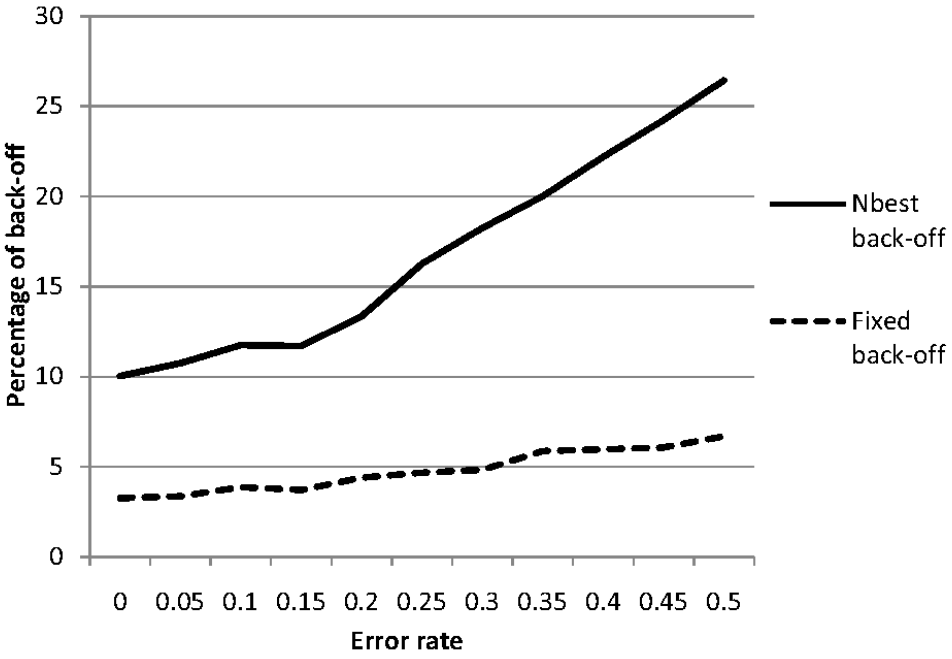

Figure 8: Percentage of backed-off actions for the Fixed back-off strategy and the N-best back-off strategy for different error rates.

tries the actions that, if valid, achieve the best reward, whereas the fixed back-off chooses the actions that rarely lead to back-off. Figure 9 shows the percentage of the first-, the second- and the third-best action taken. The results were obtained from 2500 dialogues for each error rate. It shows that on average 82% of the time the top action can be mapped to a master action, but there is a significant tail when the system backs off to the second- and the third-best action.

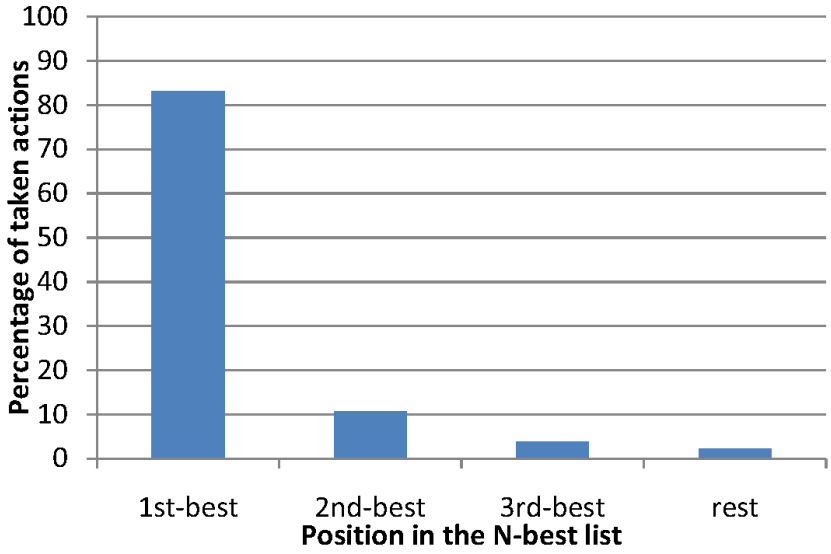

Figure 9: Percentage of actions on different positions in the N-best list taken in the N-best back-off strategy.

5.4 Natural Actor Critic algorithm

In this section, the Natural Actor Critic algorithm used in the BUDS system is evaluated. The purpose of this test is to evaluate two different sets of summary features: grid-based slot level features and entropy-based slot level features. The grid-based features discretise the marginal belief probabilities and generate seven binary features for each slot. On the other hand, the entropy-based features try to characterise the distributions of each slot. The used features are the entropy, probability of the most likely value and the second most likely value. The advantage of the entropy-based slot level features is that they are more compact. This results in less parameters to be learnt by the NAC algorithm. The NAC algorithm using grid-based features has to learn 1358 parameters while in the case of entropy-based features only 848 parameters have to be learnt.

The used training procedure is similar to the one presented in Thomson and Young (2009). The NAC algorithm executed 200 iterations and during each iteration 4000 dialogues were simulated. The training was performed on the fixed error rate of 40%. The resulting policies were evaluated on multiple error rates (confusion rates) from the range 0% to 40%. At each error rate, 5000 dialogues were simulated. The comparison of the slot level features is shown in Figure 10.

Although the results suggest that there is no statistical signifficance between the grid-based slot level features and the entropy-based slot level features, the entropy-based slot level features are preffered as they provide more compact representation of the master space. Consequently, the number of the parameters which has to be learnt by the NAC algorithm is smaller. This property might become important in more complex domains where the number of slots would limit

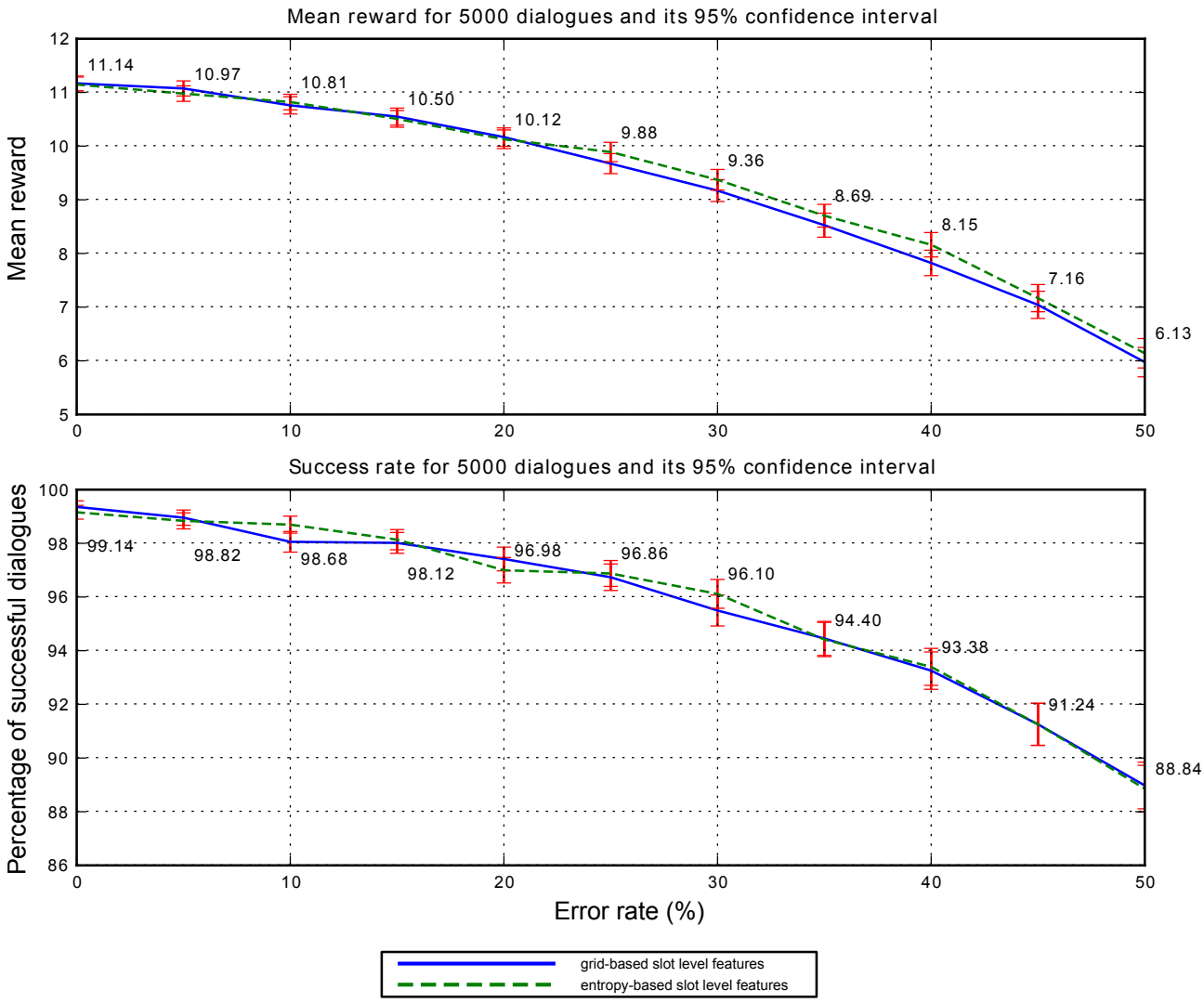

Figure 10: Comparison of the percentage of successful simulated dialogues and the mean reward between grid-based slot level features (the solid line) and entropy-based slot level features (the dashed line) on different error rates (confusion rates).

tractability of the NAC algorithm.

6 Conclusions

This report presented research into summary state/action space policy optimization for POMDP based dialogue managers.

For the HIS system, two improvements to the baseline were reported. First, k-nn Monte Carlo policy optimization (Lefèvre et al., 2009) extends the original grid-based Monte Carlo Control algorithm (Young et al., 2009) by a locally weighted k-nearest neighbor scheme which in effect smooths the decision process by interpolating the value function. The results from the evaluation with a simulated user confirmed that the k-nn policies outperform the 1-nn baseline on high noise in terms of successful dialogue completion. Second, the N-best Back-off action selection algorithm was evaluated. It was shown that invalid state-action pairs are an intrinsic problem in POMDPs. It was found that the proposed N-best Back-off action selection algorithm outperforms both the baseline with the fixed-backoff strategy and the alternative extension of the summary space.

For the BUDS system, two summary slot level features sets were evaluated. Although the results suggest that there is no statistical significance between the grid-based slot level features and the entropy-based slot level features, the entropy-based slot level features are prefered as they provide a more compact representation of the master space. Consequently, the number of the parameters which has to be learnt by the NAC algorithm is smaller.

Overall these results show that summary-based policy optimisation is effective and that both the HIS and BUDS style architectures can benefit from it. The next step is to understand how well these systems perform in real user trials. This will be the work undertaken in the final phase of the project.

References

Amari, S. (1998). Natural gradient works efficiently in learning. *Neural Computation*, 10(2):251–276.

Bishop, C. (2006). *Pattern Recognition and Machine Learning*. Springer.

Bonet, B. (2002). An e-Optimal Grid-based Algorithm for Partially Observable Markov Decision Processes. In *the Nineteenth International Conference on Machine Learning(ICML 2002),*, Sydney, Australia.

Brafman, R. I. (1997). A Heuristic Variable Grid Solution Method for POMDPs. In *AAAI*, Cambridge, MA.

Frey, B. and MacKay, D. (1998). A Revolution: Belief Propagation in Graphs With Cycles. In *In Neural Information Processing Systems*, pages 479–485. MIT Press.

Gašić, M., Lefèvre, F., Jurčíček, F., Keizer, S., Mairesse, F., Thomson, B., Yu, K., and Young, S. (2009). Back-off Action Selection in Summary Space-Based POMDP-based Dialogue Systems. In *Proc. ASRU*.

Kaelbling, L. P., Littman, M. L., and Cassandra, A. R. (1998). Planning and Acting in Partially Observable Stochastic Domains. *Artificial Intelligence*, 101:99–134.

Konda, V. and Tsitsiklis, J. (2000). Actor-critic algorithms. *Advances in Neural Information Processing Systems*, 12.

Lefèvre, F., Gašić, M., Jurčíček, F., Keizer, S., Mairesse, F., Thomson, B., Yu, K., and Young, S. (2009). k-Nearest Neighbor Monte-Carlo Control Algorithm for POMDP-based Dialogue Systems. In *Proc. SigDial 2009*.

Levin, E., Pieraccini, R., and Eckert, W. (1998). Using Markov Decision Processes for Learning Dialogue Strategies. In *Int Conf Acoustics, Speech and Signal Processing*, Seattle,USA.

Levin, E., Pieraccini, R., and Eckert, W. (2000). A Stochastic Model of Human-Machine Interaction for Learning Dialog Strategies. *IEEE Transactions on Speech and Audio Processing*, 8(1):11–23.

Littman, M. L. (1994). The Witness Algorithm: solving partially observable Markov decision processes. Technical report, Brown University.

Moore, A., Atkeson, C., and Schaal, S. (1997). Locally weighted learning. Number 11, pages 11–73.

Peters, J., Vijayakumar, S., and Schaal, S. (2005). Natural actor-critic. In *European Conference on Machine Learning (ECML)*, pages 280–291. Springer.

Pineau, J., Gordon, G., and Thrun, S. (2003). Point-based value iteration: An anytime algorithm for POMDPs. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1025 – 1032.

Schatzmann, J., Thomson, B., Weilhammer, K., Ye, H., and Young, S. (2007). Agenda-based user simulation for bootstrapping a POMDP dialogue system. In *Proceedings of Human Language Technologies / North American Chapter of the Association for Computational Linguistics (HLT/NAACL)*.

Sondik, E. J. (1971). *The Optimal Control of Partially Observable Markov Decision Processes*. Phd, Stanford University.

Sutton, R. and Barto, A. (1998). *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning. MIT Press, Cambridge, Mass.

Thomson, B., Gašić, M., Keizer, S., Mairesse, F., Schatzmann, J., Yu, K., and Young, S. (2008a). User study of the Bayesian Update of Dialogue State approach to dialogue management. In *Interspeech 2008*, Brisbane, Australia.

Thomson, B., Schatzmann, J., and Young, S. (2008b). Bayesian Update of Dialogue State for Robust Dialogue Systems. In *Int Conf Acoustics Speech and Signal Processing ICASSP*, Las Vegas.

Thomson, B. and Young, S. (2009). Bayesian update of dialogue state: A POMDP framework for spoken dialogue systems. *Computer Speech and Language*. To appear.

Williams, J. D. and Young, S. (2005). Scaling Up POMDPs for Dialog Management: The Summary POMDP Method. In *IEEE workshop on Automatic Speech Recognition and Understanding (ASRU)*, Cancun, Mexico.

Williams, J. D. and Young, S. (2007a). Partially Observable Markov Decision Processes for Spoken Dialog Systems. *Computer Speech and Language*, 21(2):393–422.

Williams, J. D. and Young, S. (2007b). Scaling POMDPs for Spoken Dialog Management. *IEEE Audio, Speech and Language Processing*, 15(7):2116–2129.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8.

Young, S. (2000). Probabilistic Methods in Spoken Dialogue Systems. *Philosophical Trans Royal Society (Series A)*, 358(1769):1389–1402.

Young, S., Gašić, M., Keizer, S., Mairesse, F., Schatzmann, J., Thomson, B., and Yu, K. (2009). The Hidden Information State Model: a practical framework for POMDP-based spoken dialogue management. *Computer Speech and Language*, 24(2):150–174.