Available online at www.sciencedirect.com
ScienceDirect
Computer Speech and Language 24 (2010) 562–588
COMPUTER SPEECH AND LANGUAGE
www.elsevier.com/locate/csl

Bayesian update of dialogue state: A POMDP framework for spoken dialogue systems

Blaise Thomson *, Steve Young

University of Cambridge, Engineering Department, Cambridge CB2 1TP, United Kingdom

Received 17 October 2008; received in revised form 19 June 2009; accepted 25 July 2009
Available online 12 August 2009

Abstract

This paper describes a statistically motivated framework for performing real-time dialogue state updates and policy learning in a spoken dialogue system. The framework is based on the partially observable Markov decision process (POMDP), which provides a well-founded, statistical model of spoken dialogue management. However, exact belief state updates in a POMDP model are computationally intractable so approximate methods must be used. This paper presents a tractable method based on the loopy belief propagation algorithm. Various simplifications are made, which improve the efficiency significantly compared to the original algorithm as well as compared to other POMDP-based dialogue state updating approaches. A second contribution of this paper is a method for learning in spoken dialogue systems which uses a component-based policy with the episodic Natural Actor Critic algorithm.

The framework proposed in this paper was tested on both simulations and in a user trial. Both indicated that using Bayesian updates of the dialogue state significantly outperforms traditional definitions of the dialogue state. Policy learning worked effectively and the learned policy outperformed all others on simulations. In user trials the learned policy was also competitive, although its optimality was less conclusive. Overall, the Bayesian update of dialogue state framework was shown to be a feasible and effective approach to building real-world POMDP-based dialogue systems.

Keywords: Dialogue systems; Robustness; POMDP; Reinforcement learning

1. Introduction

The ability to converse is often considered a defining characteristic of intelligent behaviour. Human–machine dialogue therefore has great significance in the theory of artificial intelligence. Its importance is not only academic, as evidenced by its frequent use in industrial applications where spoken dialogue systems (SDSs) are already used to access information, interact with robots, make bookings, play games and provide customer support. Any improvement in the effectiveness of these systems would have far reaching implications.

* Corresponding author. Tel.: +44 7983985454.
E-mail addresses: brmt2@eng.cam.ac.uk (B. Thomson), sjy@eng.cam.ac.uk (S. Young).

0885-2308/$ - see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.csl.2009.07.003

Commercial dialogue systems are typically implemented by charting system prompts along with possible user responses (Pieraccini and Huerta, 2008). The system is represented as a graph, sometimes called the *call flow*, where nodes represent prompts or actions to be taken by the system and the arcs give the possible responses. Formally, the system is therefore a Finite State Automaton (FSA). This approach has been the most effective commercially because prompts can be designed to elicit highly restricted user responses. However, the resulting dialogues can become frustrating for users as their choice is severely limited. Further problems arise when speech recognition and understanding errors occur. In some cases, the system might accept information that is in fact incorrect and elaborate error detection and correction schemes are required to deal with this. As a result, commercial SDSs are seldom robust to high noise levels and require significant effort and cost to develop and maintain.

Researchers have attempted to overcome these issues in various ways, many of which fall into two categories. The first is to model dialogue as a conversational game with rewards for successful performance. Optimal action choices may then be calculated automatically, reducing the costs of maintenance and design as well as increasing performance. These models have largely been based on the Markov decision process (MDP), as for example in Levin et al. (2000), Walker (2000), Scheffler (2002), Pietquin (2004) and Lemon et al. (2006). The choice of actions for a given internal system state is known as the *policy* and it is this policy which MDP models attempt to optimise.

The second avenue of research has been to use a statistical approach to model uncertainty in the dialogue. This allows for simpler and more effective methods of dealing with errors. Pulman (1996), Horvitz et al. (1999) and Meng et al. (2003) all suggest early forms of this approach. These statistical systems view the internal system state as its *beliefs* about the state of the environment. The true state of the environment is hidden and must be inferred from observations, often using Bayesian networks (BN). The environment state is often separated into different components, called *concepts*, each of which has a set of possible *concept values*. Sometimes these concepts are called *slots*.

More recently, there have been several attempts to use statistical policy learning and statistical models of uncertainty simultaneously. The resulting framework is called the Partially Observable Markov Decision Process (POMDP) (Roy et al., 2000; Williams and Young, 2006b; Bui et al., 2007). Fig. 1 shows how this framework compares to those suggested previously. This paper is concerned with the lower half of the figure; those models which use a statistical model of uncertainty.

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. Many authors suggest using Bayesian network algorithms as a solution (Pulman, 1996; Horvitz et al., 1999; Meng et al., 2003; Bui et al., 2007). Young et al. (2007) takes a different approach, grouping the state space into *partitions* where states have similar characteristics and pruning unlikely cases. Another alternative based on particle filters is suggested by Williams (2007b). However, all of these models are restricted to cases where the user's goal stays constant and there are limited dependencies between concepts or there are limited concepts or concept values.

This paper presents a method to improve standard algorithms so that belief updating becomes tractable. The technique is based on the loopy belief propagation (LBP) algorithm (Bishop, 2006, Chapter 8). LBP enables tractability by exploiting the conditional independence of concepts in the network. By grouping

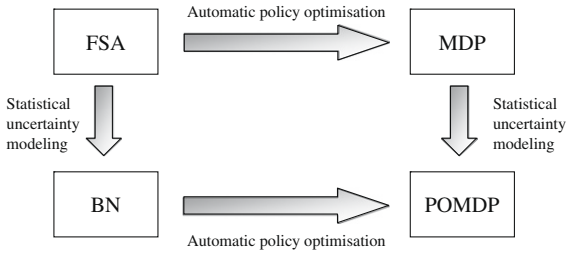

Fig. 1. Frameworks for modeling uncertainty and policy optimisation in spoken dialogue systems.

together concept values, significant improvements over the original algorithm are obtained. Further efficiency gains are obtained by assuming limited changes in the user goals. The result is then a framework which is able to cope with limited 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.

A second contribution of the paper is a new method for optimising policies in a POMDP dialogue system. Previous algorithms suggested for this task include the composite summary point based value iteration (CSPBVI) of Williams and Young (2007) and a summarised form of Q-learning presented in Thomson et al. (2007). In this paper an approach is described which utilises the Natural Actor Critic (NAC) algorithm of Peters et al. (2005) to perform policy optimisation over a factorised state space.

The NAC algorithm has several advantages over previously proposed approaches. Most importantly, the use of a component-based policy enables the system to learn to differentiate between both slot-level and higher-level actions simultaneously. In contrast, the CSPBVI algorithm learns actions for individual slots and then uses heuristics to decide which action to take. Similarly, the summarised form of Q-learning presented in Thomson et al. (2007) places much of the high-level decision into the heuristic mapping between the summary and the real-world action. Other advantages of the NAC algorithm include its ability to do online policy adaptation and its relatively strong convergence guarantees.

The paper is organised as follows: Section 2 discusses models for updating the internal system state in a spoken dialogue system. The section starts by introducing a general model of dialogue, POMDPs, Bayesian networks and other graphical models. Several network structures are particularly useful in the case of dialogue systems and these are discussed in Section 2.2. Algorithms for implementing dialogue state updates are then discussed, followed by methods to improve their efficiency in Sections 2.4 and 2.5. An efficiency analysis of the proposed algorithms is presented in Section 2.6.

Section 3 is concerned with policy optimisation, focusing on continuous-state Markov decision processes, which are introduced at the start of the section. A technique for using domain knowledge to simplify the choice of actions is presented in Section 3.2. This is followed by a discussion of the component-based policies that can be used for learning in a dialogue system. Section 3.4 then gives a detailed explanation of the Natural Actor Critic algorithm, which is the algorithm used to learn the policy parameters. This is followed by a short comparison with other learning algorithms.

An evaluation of the Bayesian updating framework is given in Section 4. The evaluation is based on both simulation results and a user trial, which are given in Sections 4.3 and 4.4 respectively. Both simulations and the user trial indicate that the Bayesian updating framework outperforms standard alternatives. Section 5 concludes the paper and gives directions for future research.

2. Maintaining dialogue state

2.1. A theoretical model for dialogue

Current models of dialogue all start from the assumption that a dialogue consists of a sequence of user and system turns. Dialogue management is concerned with deciding what to do in the system turn and this decision is always based on some form of internal state. The internal state cannot be a perfect representation of what has happened because the system does not have perfect knowledge of the environment. Instead, the internal state represents the system's *beliefs* about what has happened in the dialogue. The standard notation for the internal state varies between the POMDP and MDP literatures.[1] In this paper the system's internal state will be consistently denoted $b \in \mathcal{B}$, and is always called the *belief state*. This includes all different frameworks (including FSA and MDP) since in all cases the internal state is a representation of the system's beliefs. The belief state is often called the *system state* but that will be avoided here to stop confusion with the *environment state* defined below.

What the system chooses to do during its turn is called the *machine action* or *system action* and is chosen from a predefined set, $m \in \mathcal{M}$. The set of distributions over actions is denoted $\Pi(\mathcal{M})$. Policies, $\pi : \mathcal{B} \rightarrow \Pi(\mathcal{M})$,

[1] In the MDP literature the internal system state is denoted s. This symbol is reserved for the *environment state* in the POMDP literature.

are then mappings from belief states to probabilistic action choices. The development of the belief state over time depends on the responses the system obtains from its environment. A set of observations, $o \in \mathcal{O}$, is defined, which describes what the system can observe about the world around it.[2] Given the previous belief state, b, the last machine or system action, m, and the last observation, o, the system transitions into a new belief state, b'. The function which defines this is called the *belief state transition function*.

Clearly the actions, belief states and observations are all indexed by the turn number. When it is important to note this dependency they are denoted m_t, b_t and o_t. While the system is in state b_t it will choose action m_t according to the distribution determined by the policy, $\pi(b_t)$. The system then observes observation o_{t+1} and transitions to a new system belief state b_{t+1}. When the time dependence is insignificant, the t is omitted and a prime symbol is used to denote the next time step (e.g. $o' = o_{t+1}$).

The definition of the belief state and its transitions is what separates models into those that do or do not use a statistical model of uncertainty. The essential feature of traditional MDP and FSA models is that the number of belief states is finite. Measures of uncertainty are maintained via hand-crafted rules which represent transitions through varying levels of confidence. It is important to note that the internal state is still a representation of the system's beliefs about its environment, but the belief representation is explicitly encoded by the system designer.

In POMDP models the system-state transitions are defined indirectly via probability rules. The model assumes a set of underlying *environment states*, $s \in \mathcal{S}$. Indexed by the turn number, these are denoted s_t. Next, a stationary observation function, $P(o_t|s_t, m_{t-1})$, is assumed in which the observation probability is conditionally dependent on only the environment state, s_t, and the last machine action, m_{t-1}. Changes in the environment state are governed by a constant transition function $P(s_{t+1}|s_t, m_t)$, which assumes that changes in the environment state are dependent only on the last environment state and action but not the full state history. This is called the Markov assumption. The belief state, b_t, is the probability distribution over all possible environment states. Under the above assumptions, this distribution can be calculated using standard probability theory. It is therefore a fully statistical representation of the system's beliefs about its environment.

The difficulty with using a fully statistical approach is that belief updates quickly become intractable. A naive enumeration of all environment states grows exponentially with the number of concepts. Real-world systems must therefore use computationally efficient methods to update the belief distribution over these states.

2.2. Bayesian network models of dialogue

In order to obtain computationally efficient algorithms, the structure of the domain under consideration must be exploited. Bayesian networks provide a graphical representation of statistical models which give an intuitive representation of the structure in a system and also facilitate the use of computationally effective updating algorithms. A detailed introduction can be found in Bishop (2006, Chapter 8).

A Bayesian network is defined to be a directed acyclic graph where the nodes represent random variables, and the arcs represent conditional independence assumptions. The joint distribution of all variables in the graph factorises as the product of the conditional probability of each variable given its parents in the graph. Fig. 2 gives an example network, representing the assumptions of the POMDP model. Networks which repeat the same structure at each point in time, as in this example, are known as *dynamic Bayesian networks (DBNs)*. The sections of the network corresponding to a particular time are called *time-slices*.

The usefulness of Bayesian networks for dialogue comes by allowing further factorisations of the environment state, s_t. When factorising out components of the state, Bayesian network algorithms give efficient methods of calculating updated marginal distributions of the factors. Updating the beliefs of the system then becomes tractable.

[2] In traditional dialogue systems the actions of the system are all system prompts and the observations are the input from the recogniser. More recently, researchers have extended these ideas for the use of dialogue in troubleshooting tasks (Williams, 2007a). In these systems, actions might also include testing a user's internet connectivity or checking that their password is correct. Observations can include responses from these tests or any other perceptual input from the environment.

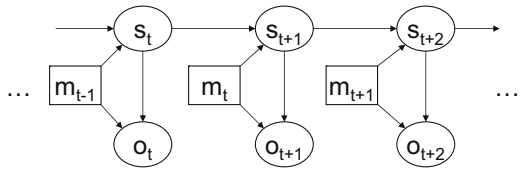

Fig. 2. A portion of a Bayesian network representing the POMDP model. Decisions are drawn in a rectangle to show that they are actions of the system rather than observed random variables.

2.2.1. Structure of dialogue networks

In a dialogue system the environment state, s_t, is highly structured. The exact structure of the Bayesian network used to describe the environment state will depend on the application but there are various structural elements which can often be reused.

One useful factorisation is to separate the environment state into three components (Williams et al., 2005). The environment state is defined as $s_t = (g_t, u_t, h_t)$, where g_t is the long term goal of the user, u_t is the true user act and h_t is the dialogue history.[3] The observed user utterance is conditionally dependent only on true user act.

In many systems, further structuring is possible by separating the state into *concepts*, $c \in \mathscr{C}$. In a tourist information system, for example, typical concepts might be the type of "food" or the "area". The state is now factorised into sub-goals, sub-user-acts and sub-histories for each concept. These are denoted $g_t^{(c)}$, $u_t^{(c)}$ and $h_t^{(c)}$. Typically $u_t^{(c)}$ is assumed to depend only on the sub-goal $g_t^{(c)}$. The sub-history $h_t^{(c)}$ will depend on the sub-user-act $u_t^{(c)}$ and the previous sub-history $h_{t-1}^{(c)}$.

It is useful to also maintain a node for the overall user act u_t. To simplify the updates, a one-to-one mapping is defined between the sub-acts, $u_t^{(c)}$ and this overall act. The sub-acts will represent parts of the action that might be relevant for a particular concept. 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. Fig. 3 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.

Validity nodes provide a simple method of switching on relevant parts of the network as the user's intention clarifies during the dialogue. For example, when a user asks about hotels in a tourist information system, the "food" concept would not be applicable. The concept might, however, be relevant when talking about restaurants in the same system. The food node would therefore be relevant if the user is looking for a restaurant and irrelevant otherwise. If it becomes clear that the user is looking for a restaurant, the validity of the food node increases and that part of the network increases in probability.

[3] In Williams et al. (2005) the notation is slightly different with $s_u = g$, $a_u = u$ and $s_d = h$.

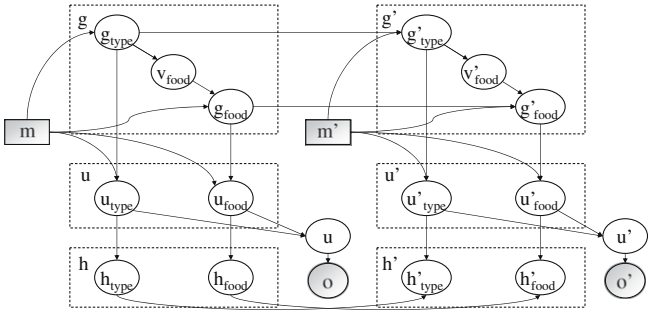

Fig. 3. An example factorisation for the Bayesian network representing part of a tourist information dialogue system. The "type" node represents the type of venue being sought and is assumed to always be relevant so v_{type} is omitted from the diagram.

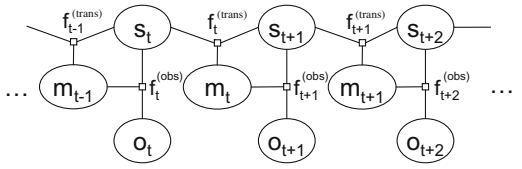

Fig. 4. A portion of a factor graph representing the POMDP model.

The remaining nodes in the network will be dependent on the application. It is important to note that the nodes need not necessarily be *slots* that are filled with *slot values*. They can represent any random variable in the system.[4]

2.3. Loopy belief propagation and factor graphs

Once a suitable structure is found to represent the complete environment state, an efficient algorithm is required to update the beliefs. When performing calculations, many algorithms are better expressed using a different graphical representation called a *factor graph* (Kschischang et al., 2001). An example factor graph is given in Fig. 4. Factor graphs are undirected bipartite graphs, with two types of node. One type represents random variables (drawn as a circle), while the other type represents factors (drawn as a small square). The assumption encoded within the graph is that the joint distribution over all random variables can be written as a product of factor functions, one for each factor node. These factors are a function of only the random variables connected to the factor node in the graph.

Since a Bayesian network defines a factorisation of the joint distribution into conditional probability factors, there is a direct mapping from Bayesian networks to factor graphs. Fig. 4 is a factor graph representation of the POMDP assumptions, previously depicted as a Bayesian network in Fig. 2. $f_t^{(trans)}$ represents the environment state's transition function, thus $f_t^{(trans)}(s_t, s_{t+1}, m_t) = P(s_{t+1}|s_t, m_t)$. $f_t^{(obs)}$ represents the observation function, $f_t^{(obs)}(s_t, o_t, m_{t-1}) = P(o_t|s_t, m_{t-1})$.

The reason for using factor graphs in dialogue management is to compute updated marginal distributions given a set of input observations (i.e. user acts). There are several possible approaches, but exact computation is generally intractable. The loopy belief propagation (LBP) algorithm gives an approximate method that is sufficiently accurate and much more efficient.

Loopy belief propagation maintains a set of messages for each arc in the model. If there is an arc between the random variable X_i and the factor f_a then two messages are defined. $\mu_{X_i \to f_a}(x_i)$ is called the message from

[4] Indeed, Williams (2007a) has discussed a Bayesian network structure for the use of POMDPs in troubleshooting tasks. Nodes in the resulting network include whether there is "no power" or "no network" for an internet user. Various different variables could be used in other applications.

the variable to the factor (into the factor) and $\mu_{f_a \to X_i}(x_i)$ is the message from the factor to the variable (out of the factor). Both of these are functions of the possible values of X_i.

The details for calculating these messages are given in Algorithm 1. An explanation of the reasoning involved in obtaining the algorithm is beyond the scope of this paper but may be found in (Bishop, 2006, Chapter 8). Once the messages are computed, the marginal probability of any variable, X_i, is calculated from the messages to that variable from the neighbouring factors, $a \in ne(X_i)$. If k is a normalising constant then:

$$p(x_i) \approx k \prod_{a \in ne(X_i)} \mu_{f_a \to X_i}(x_i). \tag{1}$$

Algorithm 1. Loopy belief propagation.

initialize: set all messages equal to one.
Let $Y = \{\mathbf{x} = (x_1, x_2, \ldots, x_N)^\top : x_i \text{ is a possible value of } X_i\}$
Note that Y enumerates the set of all combinations of the X_i random variables
repeat
Choose a factor f_a to update. Suppose this is connected to variables $X_1, X_2, \ldots, X_N$
for each variable, X_i, connected to the factor **do**
(a) Update the messages **out** of the factor

$$(\forall x_i') \quad \mu_{f_a \to X_i}(x_i') = \sum_{\mathbf{x} \in Y, x_i = x_i'} f_a(\mathbf{x}) \prod_{j \neq i} \mu_{X_j \to f_a}(x_j). \tag{2}$$

(b) Update the messages **into** nearby factors
for each factor $b \neq a$, connected to X_i **do**

$$(\forall x_i') \quad \mu_{X_i \to f_b}(x_i') = \prod_{c \neq b} \mu_{f_c \to X_i}(x_i'). \tag{3}$$

end for
end for
until convergence

The factor updates in LBP may be completed in any sequence although in practice the system designer will aim to choose a sequence which minimises the number of iterations. The updating will continue until the messages no longer change significantly. At this stage the algorithm is said to have *converged* and the resulting set of messages will constitute a *fixed point* of the algorithm. In cases where the factor graph has a tree structure, the algorithm is exact and will converge after a breadth first iteration through the tree followed by a reverse iteration. On more complex graphs, convergence is not guaranteed, although convergence does usually occur in practice.

2.3.1. Limited time-slices

One practical difficulty in using the LBP algorithm is that the number of nodes in the network will grow with time. Hence, it may be preferable to limit the number of time-slices that are maintained (Murphy, 2002). Given a number, n, the algorithm proceeds as before except that the factor updates are restricted to the most recent n time-slices. The marginal distributions for variables that are connected to factor nodes in the most recent n time-slices must still be maintained but any other information may be removed.

In the special case when only one time-slice is maintained and the single slice network has a tree structure, this approach reduces to a special case of the Boyen–Koller algorithm. No iteration will be required because of the tree structure and the error, in terms of expected KL-divergence from the exact updated distributions, will remain bounded over time as proved in Boyen and Koller (1998).

Experiments show that updating the entire history tends to give better accuracy whereas updating only the most recent time-slice gives faster computation and removes problems with convergence (Murphy, 2002). Further details on the computational efficiency of the algorithms are given in Section 2.6 and the convergence and accuracy issues are discussed in Appendix B.

2.4. Grouped loopy belief propagation

The standard LBP algorithm improves efficiency by exploiting the dependencies between concepts. In a dialogue system, a significant issue arises where a concept can take a large number of values. A tourist information system, for example, might have many thousands of options for the name of the available venues. Using loopy belief propagation will help to mitigate the effect of dependencies but updates still become intractable when there are too many concept values.

One solution to this problem is discussed in the Hidden Information State (HIS) model of Young et al. (2007). Given an observation, many of the possible environment states are indistinguishable. It therefore makes sense to join these indistinguishable states into groups. In the HIS system this is done by creating partitions of the true goal space. The problem with this is that the partitions are computed over the full unfactorised goal and so cannot exploit the conditional independencies between concepts. The approach discussed here builds on the HIS idea of partitions, but uses it in the context of Bayesian networks and factor graphs. Partitions of concept-values for a particular node are considered rather than partitions of the unfactorised goal. If observations do not distinguish between different concept-values one would expect the messages for those values to be the same.

2.4.1. An example

As an example, consider the loopy belief updates of sub-goals such as the food sub-goal in the state factorisation shown in Fig. 3. The true value of each sub-goal is $g_t^{(c)}$, while as explained in Section 2.2.1, $v_t^{(c)}$ and m_t represent the validity of the sub-goal and the machine action respectively.

An important assumption about the goal transitions is now made. Given a machine action m_t, there is some probability of staying in the same state, $\theta_{m_t,1}$ and a constant probability, $\theta_{m_t,2}$, of changing states. For the moment, the concept, c, is assumed to be applicable so $v_t^{(c)}$ is ignored. Written as an equation, the factor representing the sub-goal transition function is:

$$f_a(g_{t+1}^{(c)}, g_t^{(c)}, m_t) = P(g_{t+1}^{(c)} | g_t^{(c)}, m_t) = \begin{cases} \theta_{m_t,1} & \text{if } g_t^{(c)} = g_{t+1}^{(c)} \\ \theta_{m_t,2} & \text{if } g_t^{(c)} \neq g_{t+1}^{(c)} \end{cases}. \tag{4}$$

Eq. (2) of the LBP algorithm requires a summation over the values of this factor, fixing one value and weighting by the product of the messages from the surrounding variables, of which there is only one in this case. Since the machine action m_t is known it is also fixed. The resulting updates are as follows:

$$\mu_{f_a \to g_{t+1}}(g') = \sum_g P(g_{t+1} = g' | g_t = g, m_t) \mu_{g_t \to f_a}(g), \tag{5}$$

$$\mu_{f_a \to g_t}(g) = \sum_{g'} P(g_{t+1} = g' | g_t = g, m_t) \mu_{g_{t+1} \to f_a}(g'). \tag{6}$$

where the concept superscript (c) has been omitted for clarity.

These equations require a summation over the entire set of possible user goals. This is clearly inefficient, since when there is nothing to distinguish different goals, the messages into the factor ($\mu_{g_t \to f_a}$ and $\mu_{g_{t+1} \to f_a}$) will be constant. They can therefore be factored out of the summation, leaving a sum over the factor values which can be calculated offline. The result will be a set of equations which only require a summation for each group of user goals for which the messages are equal.

This idea is formalised by partitioning the possible user goals into groups. A collection of sets, $Z_1, Z_2, \ldots, Z_N$, is defined as a set of partitions of the values for g_t and g_{t+1} such that whenever $g \in Z_i$ and $g \in Z_j$ then $i = j$ (i.e. there is no overlap). In the example of the "food" concept, one might have three values "Chinese", "Indian" and "French". When the user has only spoken about "Chinese" food it is reasonable to split the values into two partitions – one containing only the "Chinese" option and the other containing the "Indian" and "French" options. Note that the same partitioning may be used for both the previous and the next time step since the set of values is the same.

Now let $g \in Z_i$ and $g' \in Z_j$ for some i, j and let m_t be an arbitrary machine action. The sum of the factor values for this choice of partitions may be calculated offline as follows. If $i \neq j$ then

$$\sum_{g \in Z_i} P(g_{t+1} = g' | g_t = g, m_t) = |Z_i| \theta_{m_t,2}, \tag{7}$$

and

$$\sum_{g' \in Z_j} P(g_{t+1} = g' | g_t = g, m_t) = |Z_j| \theta_{m_t,2}. \tag{8}$$

If $i = j$ then

$$\sum_{g \in Z_i} P(g_{t+1} = g' | g_t = g, m_t) = (|Z_i| - 1)\theta_{m_t,2} + \theta_{m_t,1}, \tag{9}$$

and

$$\sum_{g' \in Z_j} P(g_{t+1} = g' | g_t = g, m_t) = (|Z_j| - 1)\theta_{m_t,2} + \theta_{m_t,1}. \tag{10}$$

In all cases, the sums are independent of the particular g and g' chosen to represent the partitions. To simplify the notation, the factor value as a function of the partitions may now be defined by summing over the partition values. The partition factor value is then a simple multiple of the values calculated in Eqs. (7)–(10).

$$f_a(Z_j, Z_i, m_t) = \sum_{g \in Z_i} \sum_{g' \in Z_j} f_a(g', g, m_t) \tag{11}$$

$$= \begin{cases} |Z_i|(|Z_i| - 1)\theta_{m_t,2} + |Z_i|\theta_{m_t,1} & \text{if } i = j, \\ |Z_i||Z_j|\theta_{m_t,2} & \text{otherwise.} \end{cases} \tag{12}$$

As mentioned above, to simplify Eqs. (5) and (6) the messages into the factor must be constant before the factor values may be grouped together. Assume the messages are constant over the partition and denote the messages as a function of the partition – e.g. $\mu_{g_t \to f_a}(Z_i) = \mu_{g_t \to f_a}(g)$ for any $g \in Z_i$. The update equations can now be written as a function of the partition factor values:

$$\mu_{f_a \to g_{t+1}}(Z_j) = \sum_i \frac{1}{|Z_j|} f_a(Z_j, Z_i, m_t) \mu_{g_t \to f_a}(Z_i), \tag{13}$$

$$\mu_{f_a \to g_t}(Z_i) = \sum_j \frac{1}{|Z_i|} f_a(Z_j, Z_i, m_t) \mu_{g_{t+1} \to f_a}(Z_j). \tag{14}$$

These new equations can be used in the system by grouping together states when there is no evidence to distinguish them. The partitions are given by a collection of singletons along with a set representing the remaining options. When an observation arrives which distinguishes one of the states from the remaining options, that state is split off. Loopy belief propagation then proceeds as before, except that instead of multiplying by the normal factor value, the sums given above are used. The result is that a summation is only required for each partition rather than for every possible user goal.

An important requirement for these equations to remain valid is that the messages for each value must be equal. This means that one must use a uniform prior over the values within a partition. It may be possible to assume a non-uniform prior in certain special cases, as for example in the HIS system. An analysis of the exact circumstances where non-uniform priors may be used is left for future work.

2.4.2. General case

The use of partitions in loopy belief updates can be generalised to arbitrary factors. This allows the grouped LBP approach to be applied with validity nodes and also in the connections between the goal node and the user act. The details of the general algorithm are given in Appendix A.

2.5. Further efficiency gains

The grouped form of loopy belief propagation provides a significant reduction in calculation time. The algorithm still, however, requires an iteration over all combinations of the partitions. In certain special cases, this iteration can be simplified as well.

The case that is important for dialogue is the one discussed above in Section 2.4.1, which has the property that the factor sum is constant whenever the partitions, Z_i and Z_j, are different. One can therefore factor out this term from Eq. (13) to give:

$$\mu_{f_a \to g_{t+1}}(Z_j) = \frac{1}{|Z_j|} \sum_i \mu_{g_t \to f_a}(Z_i) f_a(Z_j, Z_i, m_t) = ((|Z_j| - 1)\theta_{m_t,2} + \theta_{m_t,1})\mu_{g_t \to f_a}(Z_j) + \theta_{m_t,2} \sum_{i \neq j} |Z_i| \mu_{g_t \to f_a}(Z_i).$$

In loopy belief propagation, the effect of the messages on the estimated marginal distributions is independent of their scale and it is often necessary to rescale messages to avoid numerical issues. This can be exploited to further simplify the update equation. For grouped LBP, a suitable rescaling is to set:

$$\sum_i |Z_i| \mu_{g_t \to f_a}(Z_i) = 1. \tag{15}$$

Once the messages are scaled in this way, the update becomes:

$$\begin{aligned} \mu_{f_a \to g_{t+1}}(Z_j) &= ((|Z_j| - 1)\theta_{m_t,2} + \theta_{m_t,1})\mu_{g_t \to f_a}(Z_j) + \theta_{m_t,2}(1 - |Z_j| \mu_{g_t \to f_a}(Z_j)) \\ &= \theta_{m_t,1} \mu_{g_t \to f_a}(Z_j) + \theta_{m_t,2}(1 - \mu_{g_t \to f_a}(Z_j)). \end{aligned}$$

This update can therefore be done without any iteration through combinations of partitions. Instead, one only has to iterate through the set of partitions, resulting in a large reduction in computation. The update for the messages to the previous state can be optimised in a similar way.

2.6. Efficiency analysis

To formally analyse the framework's efficiency, consider the updates for a factor with K different dependent variables, each having N values. Suppose further that the values for each node may be split into P indistinguishable partitions. The grouped form of loopy belief propagation must iterate through all combinations of the partitions and will therefore take $\mathcal{O}(P^K)$ calculations. Some extra partitioning machinery is required to ensure that enough partitions are included. This requires only a single pass through the current list of partitions so does not affect the complexity significantly. When the values are not grouped, the same factor updates would require $\mathcal{O}(N^K)$ calculations.

When a constant probability of change is assumed, there is no longer the need to go through all the combinations of the variable values. When values are grouped this results in a decrease by a factor of P. Goal updates in the model will therefore take $\mathcal{O}(P)$ calculations rather than $\mathcal{O}(P^2)$.

2.6.1. Experimental evaluation of efficiency

An experimental evaluation of the algorithms' efficiency was performed by testing the computation times on several simple models.[5] A detailed description of the models is given in Appendix B.1. Four algorithms were compared:

- The one time-slice LBP algorithm (1 bp1). This is equivalent to the Boyen–Koller algorithm.
- The one time-slice LBP algorithm assuming a constant probability of change (cnst).
- The one time-slice LBP algorithm grouping values (grp).
- The one time-slice LBP algorithm assuming a constant probability of change and grouping values (cnst-grp).

[5] All experiments were performed on an Intel Core 2 Quad CPU, 2.4 GHz processor with 4 GB of RAM.

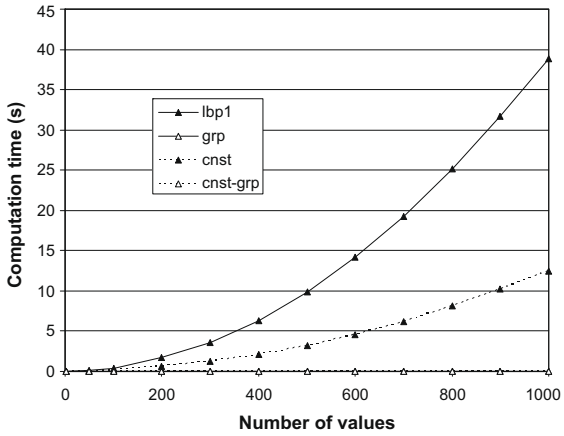

Fig. 5. Computation times of various algorithms for the marginal distribution after 10 time-slices in a Bayesian network with 1 node. Further details of the network are given in Appendix B.1. About 500 samples were drawn from the network and the mean calculation time is plotted. The two algorithms using grouping take a maximum computation time of 0.04 s and are therefore indistinguishable on the graph. The standard errors of all estimates are below 0.01 s.

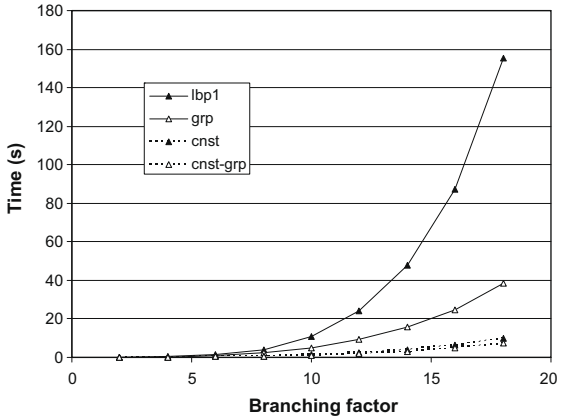

Fig. 6. Computation times of various algorithms for 10 time-slices of a Bayesian network. The network has a tree structure of depth 2 with varying branching factors. Further details of the network are given in Appendix B.1. About 500 samples from each network were drawn and the mean calculation times are plotted. The standard errors of all estimates are below 0.7 s. The standard errors for estimates of the *cnst* and *cnst-grp* algorithms are below 0.07 s.

The first experiment used a model with a single concept ($K = 1$) and the results are given in Fig. 5. The computation times clearly follow the trends suggested by the theory above. As the number of values increases, the computation time of the standard algorithm increases with the square of the number of values. Assuming a constant probability of change reduces the computation time to a linear increase. Grouping values dramatically improves performance time as so few partitions are required that there is little computation to be done.

Fig. 6 shows an evaluation of the proposed algorithms on models with more dependencies. In this figure, a tree structure of depth 2 and varying branching factors is used to represent the hidden goal. This gives an abstraction of the hierarchical dependencies often present in dialogue systems. Higher-level node values in this network can be grouped only rarely since there are too many other nodes which depend on them. One can see how this results in the assumption of a constant probability of goal changes being more effective than grouping. Grouping does, however, still improve performance.

These results show how the two techniques of grouping and assuming constant changes work together to improve efficiency. Grouping reduces the computation due to large numbers of values for a particular node.

Assuming constant changes in the goal helps to reduce the computation required for nodes with more dependencies.

The factors connecting the sub-goals to the sub-acts can also use the grouped form of LBP by ensuring the partitions have equal user act probabilities. In most cases, this will not be a significant assumption. For example, it is reasonable to assume that the probability of the user saying "I would like Chinese food" given that they want Indian or Italian food would be the same.

Once all possible simplifications have been included into a dialogue system, all of the important factor updates will require linear time in the number of concept-level partitions. The connection between the sub-acts and the overall act is deterministic so it requires only an iteration through the overall act values. The factors connecting history nodes have very few values to iterate through and do not contribute significantly to computation time. Further dependencies can also be modeled if necessary, but will bring associated increases in complexity.

2.6.2. Comparison with other approaches

When the unfactorised goal is used for belief updating then the number of partitions at the slot-level must be multiplied to obtain the number of unfactorised goal partitions. If the system state contains P partitions for each of K slots, the Hidden Information State model will require $\mathcal{O}(P^K)$ calculations to do the update.

$\mathcal{O}(P^K)$ is the same computation time as the proposed algorithms with all dependencies included. The difference is that belief updates in the HIS model are calculated exactly on the assumption that user goals do not change. The algorithms presented in this paper allow for changes in the user goal but necessitate that approximations be made. If the entire goal is considered as a single node then the calculation times and accuracy are essentially the same. One can therefore think of the grouped belief propagation algorithm as an extension of the HIS approach to systems with conditionally independent slots and changes in the user goal.

An alternative approach to updating the belief state is to use particle filters (Williams, 2007b). Comparing this method with the approach here is difficult as the particle filter's computation time depends crucially on the number of particles used, which, for a given level of accuracy, is highly problem dependent. A fair comparison is therefore left for future work.

3. Policy design

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, $b \in \mathcal{B}$, to probabilistic action choices. The structure of the policy will depend on the form of the belief state b.

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 of Section 2. The policy then maps these updated beliefs to system actions. This policy can be hand-crafted, using decision trees based on the various probabilities. Horvitz et al. (1999) have suggested the Value of Information as a suitable quantity for basing these hand-crafted decisions.

Hand-crafting policies is time consuming and can result in sub-optimal decisions. Various researchers have attempted to overcome these problems by learning policies automatically. Automatic policy learning is the subject of the rest of this section.

3.1. Markov decision processes

The model traditionally used for policy learning is a Markov decision process (MDP), in which the belief state sequence, $b_0, b_1, \ldots, b_T$, is finite and Markov.[6] The feature of an MDP which enables learning is the reward function, R, which defines the immediate reward for being in a particular state and taking a particular

[6] The system is Markov when the probability of b_{n+1} given b_n is conditionally independent of b_i, for all $i < n$. The system is said to be *episodic* when the belief state sequence is finite. Non-episodic MDPs also require the definition of a discount rate, γ, which will be omitted here.

action, m. The aim of policy learning is then to optimise the expected future reward. This expected future reward is known as the *value function* which depends on the policy, π, and the start state, b:

$$V^{\pi}(b) = \sum_{m} \pi(b,m)R(b,m) + \sum_{m} \sum_{b'} \pi(b,m)P(b'|b,m)V^{\pi}(b'). \tag{16}$$

Several other quantities are useful when working with MDPs. The Q-function, $Q^{\pi}(b,m)$, is the expected future reward obtained by starting with a particular action and then following the policy. The advantage function, $A^{\pi}(b,m)$, is the difference between the Q-function and the value function. The occupancy frequency, $d^{\pi}(b)$, gives the expected number of times each state is visited.[7] These three quantities are given by the following equations:

$$Q^{\pi}(b,m) = R(b,m) + \sum_{b'} P(b'|b,m)V^{\pi}(b'), \tag{17}$$

$$A^{\pi}(b,m) = Q^{\pi}(b,m) - V^{\pi}(b), \tag{18}$$

$$d^{\pi}(b) = \sum_{t=0}^{\infty} P(b_t = b). \tag{19}$$

In the case of a POMDP, the reward is traditionally written as a function of the environment state s rather than the belief state b. The Markov assumption is also usually made on the environment states rather than the belief states. Fortunately, one can show that if the environment states are Markov then the belief states are too. A belief state reward may be derived from the environment state reward as follows:

$$R(b,m) = \sum_{s \in \mathcal{S}} b(s)R(s,m). \tag{20}$$

This new reward can be used to define a continuous-state MDP that is closely related to the original POMDP. The internal state space of this MDP is identical to the belief state of the POMDP and policies will optimise the MDP only if they also optimise the original POMDP (Kaelbling et al., 1998). Throughout this paper, optimisation is performed on this belief state MDP instead of the original POMDP. This allows the techniques to be generalised for use in both MDP and POMDP frameworks, or even in hybrid approaches.

For the purpose of learning, it is useful to consider policies that are parameterised according to some parameter vector $\boldsymbol{\theta}$. The policy gives the probability of an action depending on the parameters and the current belief state, $\pi(m|b,\boldsymbol{\theta})$. Policy learning can then be achieved by performing gradient descent on the policy whilst attempting to optimise $V^{\pi}(b)$.

Before detailing the calculations, two techniques will be described for simplifying learning in a POMDP dialogue system. Firstly, the action set can be significantly reduced by encoding expert knowledge into summary actions, as shown in Section 3.2. Secondly, a parameterised policy can be used to factorise the effects of different concepts, as explained in Section 3.3.

3.2. Summary actions

In a dialogue system there are many cases when it is obvious to a system designer that one action is better than another. 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".

Summary actions exploit this idea to significantly reduce the size of the action set. Along with the original machine action set, $\widetilde{\mathcal{M}}$, a set of summary actions, $\mathcal{M}$, is defined. Note that $\mathcal{M}$ is used here as the summary action set rather than the original set, as in Section 2, because it is the set of actions used for learning. Each

[7] This is known to be finite because the system is episodic.

summary action, m, is defined by the system designer and represents a subset of $\widetilde{\mathcal{M}}$. Given a summary action, m, and belief state, $\mathbf{b}$, a mapping back to the original action set, $F(m, \mathbf{b})$, is defined. In the example, "confirm food" action would be the associated summary act and given a belief state, the corresponding machine act in $\widetilde{\mathcal{M}}$ would be to confirm the most likely food type.

A new system can be defined using this summarised action set. The belief states are exactly as before but the actions are restricted to $\mathcal{M}$. Given a summary action and belief state, m and $\mathbf{b}$, the system simply takes the corresponding original action, $F(m, \mathbf{b})$. Rewards and transitions are defined as before using this corresponding action. The system is still an MDP, but the action set is reduced.

This process of using summary actions allows the system designer to encode expert knowledge to allow learning to proceed more quickly. The optimal policy for this summarised version of the MDP will be close to the optimal policy for the full MDP provided that the inverse mapping, F, is reasonably accurate.

The use of summary actions is based on the summary POMDP idea proposed by Williams et al. (2005). The difference is that the summary POMDP factors both the state and actions according to a set of slots. This necessitates a calculation of the state dynamics at a slot-level and makes learning between actions that affect different slots impossible. Also, the changed dynamics of the summary POMDP require special learning algorithms. In contrast, the approach here does not change the state, makes no change to the state dynamics and preserves the properties of a standard MDP so no special techniques are required.

3.3. Component-based policies for dialogue

When automating decision making, it is useful to be able to separate out the effects of different components. Using parameterised policies provides a simple method of implementing this.

A standard parameterisation for the policy is to use a softmax function. For each summary machine action, m, a *basis function* is defined, $\boldsymbol{\phi}_m$, which is a vector function of the belief state, $\mathbf{b}$. The basis function gives a set of features from the belief state that are important for decision making, using any arbitrary function for each element. Along with the parameters, $\boldsymbol{\theta}$, these functions determine the likelihood of taking an action according to the following equation:

$$\pi(m|\mathbf{b}, \boldsymbol{\theta}) = \frac{e^{\boldsymbol{\theta}\cdot\boldsymbol{\phi}_m(\mathbf{b})}}{\sum_{m'} e^{\boldsymbol{\theta}\cdot\boldsymbol{\phi}_{m'}(\mathbf{b})}}. \tag{21}$$

This parameterisation provides for a simple method to separate the effects of each component on the overall probability of taking each action. Each concept, $c \in \mathcal{C}$, is associated with a concept-level basis function, $\boldsymbol{\phi}_{m,c}(\mathbf{b})$. Extra information is encoded in an overall basis function, $\boldsymbol{\phi}_{m,*}(\mathbf{b})$. The full basis function can then be separated into components as follows:

$$\boldsymbol{\phi}_m(\mathbf{b})^\top = [\boldsymbol{\phi}_{m,1}(\mathbf{b})^\top, \ldots, \boldsymbol{\phi}_{m,K}(\mathbf{b})^\top, \boldsymbol{\phi}_{m,*}(\mathbf{b})^\top]. \tag{22}$$

Each concept-level function must encode a set of features that depends on the machine action. As an example, a typical "slot-filling" dialogue might separate $\boldsymbol{\phi}_{m,c}(\mathbf{b})$ into components which each depend on how the machine action affects the given slot, c. A straightforward way of achieving this is to use an indicator function along with a set of sub-components. For example, if a set of N_c different sub-components are defined then $\boldsymbol{\phi}_{m,c}(\mathbf{b})^\top = [\boldsymbol{\phi}_{m,c,1}(\mathbf{b})^\top, \ldots, \boldsymbol{\phi}_{m,c,N_c}(\mathbf{b})^\top]$, where all of these *sub basis functions*, $\boldsymbol{\phi}_{m,c,i}(\mathbf{b})$, are $\mathbf{0}$ except for the sub-component corresponding to the given machine action. This nonzero component will be a function of the beliefs, decided by the system designer. The component should give an indication of the effect of the beliefs on action decisions for that particular slot.

As an example, consider a system where the only available actions are to request the value of different slots. The system designer may decide that the only important information for policy making is the probability of the most likely value for each concept, which is denoted l_c. This probability is to be multiplied by one parameter, $\theta_{c,1}$, if the action is to request this slot and a different parameter, $\theta_{c,2}$, if the action is to request a different slot. A suitable set of functions is then:

$$\boldsymbol{\phi}_{m,c}(\mathbf{b}) = \begin{cases} \begin{pmatrix} l_c \\ 0 \end{pmatrix} & \text{if } m \text{ requests slot } c, \\ \begin{pmatrix} 0 \\ l_c \end{pmatrix} & \text{otherwise} \end{cases}. \tag{23}$$

In practice, a grid-based approach may well be preferred for these sub basis functions. This is the approach used later in this paper, where a value of 1 is assigned to the component of the grid given by the beliefs and a value of 0 is assigned to all other components. For example, the probabilities of the two most likely slot values might be formed into a tuple with tuples grouped together whenever the most likely slot value is less than 0.5 and otherwise identified with the closest point in the set {(1.0, 0.0), (0.8, 0.0), (0.8, 0.2), (0.6, 0.0), (0.6, 0.2), (0.6, 0.4)}. A given sub basis function, $\boldsymbol{\phi}_{m,c,i}(\mathbf{b})$, will then map to a 7-dimensional vector with 1 assigned to the dimension associated with the corresponding group and 0 everywhere else. Other alternatives are possible, for example using two components: the entropy of the slot and the probability of the most likely option.

The final basis function that must be defined is the overall basis function $\boldsymbol{\phi}_{m,*}(\mathbf{b})$. This function allows the system designer to incorporate knowledge about the overall state in the decision making, for example, the number of database items that match a given request. For the current most likely user goal, the number of items is determined and grouped into one of "no venues", "one venue", "a few venues" or "many venues". $\boldsymbol{\phi}_{m,*}(\mathbf{b})$ is then a 4-dimensional vector with 1 in the dimension associated with the relevant group and 0 otherwise. Policy learning is used to decide how important this information is. Other knowledge about the overall state of the system could be incorporated in a similar way.

3.4. Natural Actor Critic

Once the policy has been parameterised using a suitable structure, the system must learn the parameters that optimise the expected future reward. There are various methods for doing this but one that has worked well in the framework presented here is the Natural Actor Critic (NAC) algorithm (Peters et al., 2005), which performs policy optimisation using a modified form of gradient descent.

Traditional gradient descent iteratively adds a multiple of the gradient to the parameters being estimated. This is not necessarily the most effective gradient to use because the Euclidean metric may not be appropriate as a measure of distance. In general, the parameter space is described as a Riemann space, and a metric tensor, $G_{\boldsymbol{\theta}}$, can be defined such that for small changes in the parameters $\boldsymbol{\theta}$, the distance is $|d\boldsymbol{\theta}|^2 = d\boldsymbol{\theta}^\top G_{\boldsymbol{\theta}}\, d\boldsymbol{\theta}$. The traditional Euclidean distance is then given by a metric tensor equal to the identity matrix. In optimising an arbitrary loss function, $L(\boldsymbol{\theta})$, Amari (1998) shows that for a general Riemann space the direction of steepest descent is in fact $G_{\boldsymbol{\theta}}^{-1} \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})$. This gradient is known as the *natural gradient* to contrast it with the *vanilla gradient* traditionally used.

The optimal metric tensor to use in a statistical model is typically the Fisher Information Matrix which has been shown to give distances that are invariant to the scale of the parameters (Amari, 1998). Given a probability distribution $p(x|\boldsymbol{\theta})$, the Fisher Information is the matrix $G_{\boldsymbol{\theta}}$ such that $(G_{\boldsymbol{\theta}})_{ij} = \mathbb{E}\left(\frac{\partial \log p(x|\boldsymbol{\theta})}{\partial \theta_i} \frac{\partial \log p(x|\boldsymbol{\theta})}{\partial \theta_j}\right)$. Peters et al. (2005) shows that the Fisher Information matrix for an MDP model is:

$$G_{\boldsymbol{\theta}} = \int_{\mathcal{B}} d^{\pi}(b) \int_{\mathcal{M}} \pi(m|b,\boldsymbol{\theta}) \nabla_{\boldsymbol{\theta}} \log \pi(m|b,\boldsymbol{\theta}) \nabla_{\boldsymbol{\theta}} \log \pi(m|b,\boldsymbol{\theta})^\top \, dm\, db. \tag{24}$$

The direction of steepest descent is the inverse of this matrix multiplied by the traditional vanilla gradient. The vanilla gradient for an MDP is given by the Policy Gradient Theorem of Sutton et al. (2000):

$$\nabla_{\boldsymbol{\theta}} V(b_0) = \int_{\mathcal{B}} d^{\pi}(b) \int_{\mathcal{M}} A^{\pi}(b,m) \pi(m|b,\boldsymbol{\theta}) \nabla_{\boldsymbol{\theta}} \log \pi(m|b,\boldsymbol{\theta})\, dm\, db. \tag{25}$$

Eq. (25) depends crucially on two quantities which are difficult to compute: the advantage function, defined in Eq. (18), and the occupancy frequency,[8] $d^\pi(b) = \sum_{t=0}^{\infty} P(b_t = b)$. The process of integrating over the occupancy frequency may be approximated with sampling techniques as is shown in Section 3.4.1. At the same time, the advantage function is replaced by a compatible approximation. The resulting equations for the natural gradient then simplify dramatically since the Fisher Information matrix cancels. More specifically, Peters et al. (2005) shows that if an approximate advantage function, $a_{\mathbf{w}}(b,m)$ is chosen such that:

$$a_{\mathbf{w}}(b,m) = \bigtriangledown_{\boldsymbol{\theta}} \log \pi(m|b,\boldsymbol{\theta}) \cdot \mathbf{w}, \tag{26}$$

where $\mathbf{w}$ minimises the average squared approximation error, i.e.

$$\frac{\partial}{\partial \mathbf{w}} \int_{\mathscr{B}} d^\pi(b) \int_{\mathscr{M}} \pi(m|b,\boldsymbol{\theta}) \left(A^\pi(b,m) - a_{\mathbf{w}}(b,m)\right)^2 dm\, db = \mathbf{0}, \tag{27}$$

then the required natural gradient is given by

$$G_{\boldsymbol{\theta}}^{-1} \bigtriangledown_{\boldsymbol{\theta}} V(b_0) = \mathbf{w}. \tag{28}$$

This gradient can be used to develop an algorithm with relatively strong convergence properties. The system iterates between evaluating the policy and improving it. The evaluation, called the *critic* step, must estimate the approximate advantage function. Once a suitable function is estimated an *actor improvement* is made by changing the parameters by a multiple of the natural gradient. The algorithm, called the Natural Actor Critic (NAC), is sure to converge to a local maximum of the value function as long as Eq. (27) is satisfied at every step and certain minor conditions are met.

3.4.1. Sampling methods

Eq. (27) requires a minimisation which does not generally have an analytical solution. As a result, the NAC algorithm uses sampling to *estimate* the gradient. Assuming that the dialogues are numbered $n = 1 \ldots N$ and the turns are numbered $0, \ldots, T_n - 1$, the equivalent sampled equation to minimise is:

$$\sum_{n=1}^{N} \sum_{t=0}^{T_n-1} \left|A(b_{n,t}, m_{n,t}) - a_{\mathbf{w}}(b_{n,t}, m_{n,t})\right|^2. \tag{29}$$

Since the system does not actually have estimates for the true advantage function, $A(b,m)$, the rewards in a dialogue must be grouped together in order to obtain suitable estimates. The sum of the rewards gives an unbiased estimate of the sum of the advantages and the initial value function. The resulting equation is as follows:

$$\sum_{t=0}^{T_n-1} A(b_{n,t}, m_{n,t}) + V^\pi(b_0) \approx \sum_{t=0}^{T_n-1} r(b_{n,t}, m_{n,t}). \tag{30}$$

The task of solving for a suitable approximation is now changed from minimising Eq. (29) to minimising the average error per dialogue. If J represents an estimate for the value function of the start state then the error is:

$$\sum_{n=1}^{N} \left| \sum_{t=0}^{T_n-1} A(b_{n,t}, m_{n,t}) - \sum_{t=0}^{T_n-1} a_{\mathbf{w}}(b_{n,t}, m_{n,t}) \right|^2, \tag{31}$$

$$= \sum_{n=1}^{N} \left| \sum_{t=0}^{T_n-1} r(b_{n,t}, m_{n,t}) - \left(\sum_{t=0}^{T_n-1} \bigtriangledown_{\boldsymbol{\theta}} \log \pi(m_{n,t}|b_{n,t}, \boldsymbol{\theta}) \right) \cdot \mathbf{w} - J \right|^2. \tag{32}$$

This is a simple least squares regression problem and can be solved using standard techniques. When the policy parameters are changed in an actor improvement step, the previous dialogues no longer give valid estimates for the approximation. In order to reduce this effect, previous dialogues are successively deweighted. Algorithm 2 gives a complete implementation of episodic NAC.

[8] The occupancy frequency is also sometimes called the *state distribution* (Peters et al., 2005).

Algorithm 2. Episodic Natural Actor Critic

for each dialogue, n **do**
Execute the dialogue according to the current policy π.
Obtain the sequence of states, $b_{n,t}$, and machine actions, $m_{n,t}$
Compute statistics for the dialogue:

$$\boldsymbol{\psi}_n = \left[\sum_{t=0}^{T_n} \nabla_{\boldsymbol{\theta}} \log \pi(m_{n,t}|b_{n,t}, \boldsymbol{\theta})^\top, 1 \right]^\top,$$

$$R_n = \sum_{t=0}^{T_n} r(b_{n,t}, m_{n,t}).$$

Critic Evaluation: Choose $\mathbf{w}$ to minimise $\sum_n (\boldsymbol{\psi}_n^\top \mathbf{w} - R_n)^2$
Actor-Update: If $\mathbf{w}$ has converged:
Update the policy parameters: $\boldsymbol{\theta}_{n+1} = \boldsymbol{\theta}_n + \mathbf{w}_0$, where $\mathbf{w}^\top = [\mathbf{w}_0^\top, J]$
Deweight all previous dialogues: $R_i \leftarrow \gamma R_i$, $\boldsymbol{\psi}_i \leftarrow \gamma \boldsymbol{\psi}_i$ for all $i \leqslant n$
end for

3.5. Comparison with other algorithms

The natural actor critic is one of several algorithms that could be used to learn the policy. Other popular learning algorithms include TD-learning, SARSA-learning and various improvements of point-based value iteration (Sutton and Barto, 1998; Shani et al., 2008; Williams and Young, 2006).

One particularly important characteristic of the NAC algorithm is that it is *model-free* rather than *model-based*. This means that the algorithm learns from sample dialogues rather than using the model inherent in the Bayesian network. The major advantage of this is that it is much easier to build a good simulator of the environment than it is to build a good probabilistic model (Sutton and Barto, 1998). Another important reason for using a model-free approach is that the observations in a dialogue system are not finite. The use of confidence scores requires a continuous set of observations, unless they are discretised. This is the main reason for using NAC over model-based techniques such as Point Based Value Iteration.

NAC's strong convergence properties are the main reason it should be preferred over other model-free approaches, such as TD- and SARSA learning. The use of gradient descent results in small changes to the policy. This allows for a much more gradual learning process, which reduces the chances of reaching a local optimum (Peters et al., 2005).

4. Evaluation

4.1. An example task

The framework described above was evaluated by comparing it with a conventional MDP system for the tourist information domain, in which users may ask about hotels, restaurants, bars and amenities in a fictitious town. The dialogue is mixed-initiative, meaning that the system may ask specific questions while users may provide information that was not requested or ask their own questions. Four example systems were built in order to compare differences between hand-crafted and learned policies as well as the effects of using a statistical approach to model uncertainty (i.e. the four frameworks of Fig. 1).

The user may speak about nine concepts in their attempts to find a suitable venue. These are: name of the venue, type of venue, area, price range, nearness to a particular location, type of drinks, food type (for restaurants), number of stars (for hotels and restaurants) and music (for restaurants and bars). Once a suitable

venue is found the user may ask about four further concepts: address, telephone number, a comment on the venue and the price (for hotels and restaurants). The database contains 47 different venues.

A simulated user was built for the task using the agenda-based approach of Schatzmann et al. (2007). The simulated user receives the semantic level output of the dialogue system and gives a response deemed to reflect the way a human user would behave. The simulated user is a useful tool for developing dialogue systems as it supports both policy learning and evaluation. The same simulator was used for both learning and for evaluation.

Outputs of the simulated user were passed through a simulated error channel before being input to the dialogue manager. The error channel makes use of a *confusion rate*, ρ, which determines the amount of error produced. At each turn, three alternatives are produced which are each correct with probability $1 - \rho$. These alternatives are then collated into an N-best list of user acts and confidence scores are attached. Most experiments used a confusion rate of 40%. A typical run of 1000 dialogues at this confusion rate results in a top accuracy of 80.9, an oracle accuracy of 96.3 and an ICE score of 0.96. The top accuracy gives a measure of the usefulness of the most likely hypothesis, the oracle accuracy gives the accuracy for the best hypothesis in the N-best list and the ICE score measures the overall usefulness of the confidence scores. Further details on these metrics may be found in Thomson et al., 2008.

Evaluation was based on four metrics:

- *Reward:* the reward is 20 less the number of dialogue turns for a successful dialogue and 0 less the number of turns for an unsuccessful one. A dialogue was considered successful if a suitable venue was offered and all further pieces of information were given. In the case where no venue matched the constraints, the dialogue was deemed successful if the system told the user that no venue matched and a suitable alternative was offered.
- *Objective success:* a dialogue was objectively successful if a suitable venue was offered and all further pieces of information were given. If no venue matched the constraints then the dialogue was successful if the user was told that there was no matching venue
- *Objective score:* if the dialogue was objectively successful, then the score was 20 less the number of turns until success. Otherwise, it was 0 less the number of dialogue turns.
- *Subjective success:* for testing with real users only, a dialogue was deemed subjectively successful if the user believed that their requirements had been satisfied.

The difference in the definitions of success for the reward and the objective score is due to a lack of information when testing with human users. From the dialogue transcriptions, it is not possible to tell how a human user changes their goals when told there is no matching venue whereas the simulated user provides this information explicitly.

4.2. The four systems

A Bayesian network for the tourist information domain was built using the framework for Bayesian updates of dialogue state (BUDS) discussed in Section 2.2. A user goal node, a user act node and a history node were implemented for each of the nine concepts used to find suitable venues. User act nodes and history nodes were added for the four extra concepts that could be asked. An extra "method" node was also included to capture the approach being used by the user to find the venue. This enables a venue to be found by name,

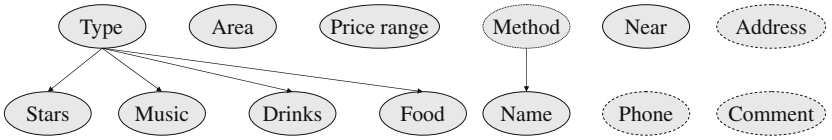

Fig. 7. A graphical representation of the concepts for one time-slice in the BUDS experimental system. The "phone", "address" and "comment" concepts are ignored at the goal-level and only maintain nodes at the act- and history-levels. The other concepts have corresponding nodes at the goal level with a Bayesian network structure as depicted in the diagram.

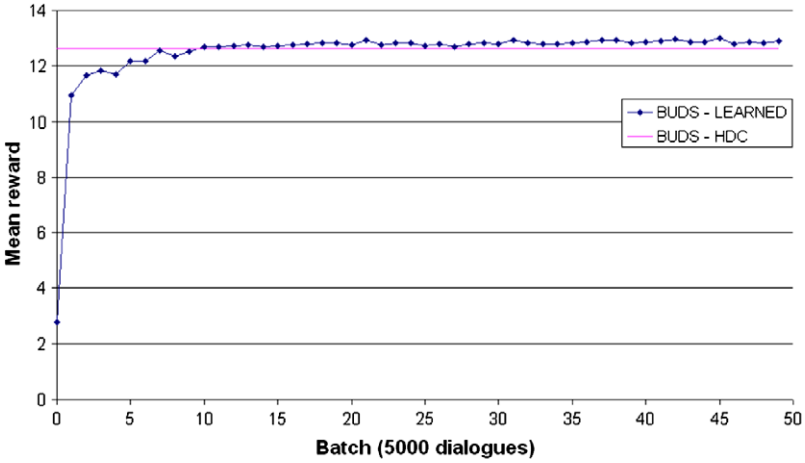

Fig. 8. Average reward over 5000 dialogues during training of the policy for use with the BUDS framework. The horizontal line gives the reward obtained by the hand-crafted policy with the same model for updating the belief state (BUDS–HDC).

according to a set of constraints or for a set of alternative venues to be requested given some constraints. Fig. 7 shows a graphical representation of the concepts in the system and their dependencies.

Policy learning for the POMDP system was implemented using the episodic Natural Actor Critic algorithm and a component-based policy, as explained in Section 3. One summary action enabled the system to offer information about a venue while the others asked the user to confirm, request or select one of two options for each of the different concepts.

Although the NAC algorithm can be performed online, online learning with real humans would require too much effort. Instead, the system interacted with the agenda-based simulator discussed above. The same reward function was used as for evaluation, thus a 20 point reward was given for full completion of a dialogue and 1 point was subtracted for each dialogue turn. Fig. 8 shows how the average reward improves as more dialogues are completed. After 250,000 dialogues the policy was fixed, giving a learned policy for the proposed framework (BUDS–TRA).[9]

A hand-crafted policy for the BUDS framework was implemented by building up overall actions from sub-actions for each concept (labeled BUDS–HDC). All slots where the most likely value had probability greater than 0.8 were accepted. When the accepted slots gave sufficient information for a venue recommendation, the system gave the information. Otherwise the system would implicitly confirm, explicitly confirm, ask the user to select between two options or request (in that order) one of the low probability slots. In cases where the most likely value had probability greater than 0.5, the slot was confirmed or implicitly confirmed. When the probability was less than 0.4 it was requested. Otherwise the user was asked to select between the two most likely options.

Comparisons were made with two MDP systems which had a belief state composed of the same concepts as used in the BUDS-based systems. Each concept or slot could take three state values: unknown, known or confirmed. State transitions were based only on the most likely user act since it is impractical to use a full N-best list of user acts in an MDP system. No information from the confidence scores was maintained. A hand-crafted policy was built to either offer information or request or confirm a slot based on this state (this system is labeled HDC). A policy using the same belief state was also trained using standard reinforcement learning techniques (labeled MDP). These MDP systems were built by another researcher in another project and considerable effort had been expended to make them as competitive as possible (Schatzmann, 2008).

[9] The error model for training used a confusion rate of 0.4. This was not changed for different tests, ensuring that the system was not aware of the noise conditions in which it was tested.

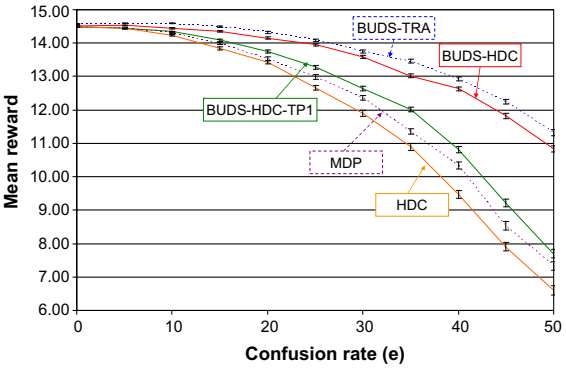

Fig. 9. Simulated comparison of the four frameworks for dialogue management. Each point gives the mean reward for 5000 simulated dialogue. Error bars show one standard error on each side of the mean.

4.3. Simulation results

Fig. 9 shows the results of a simulated comparison of the four systems. In addition, a variant of the BUDS based system with the hand-crafted policy was also tested (BUDS–HDC-TP1) which used only the most likely output from the simulated error channel. This allowed the benefit of using the full N-best semantic output to assessed.

The figure highlights some important features of the different policies. At low error rates all policies perform equally well, but at higher error rates a consistent ordering of the approaches emerges. Firstly, the two systems that make use of the full N-best list of user acts show significantly better performance. The importance of the N-best list is particularly evident in the comparison between the BUDS–HDC and BUDS–HDC-TP1 systems, which use all the same components except for the size of the N-best list. Secondly, the use of a statistical model of uncertainty outperforms hand-crafted models of the belief state, even when only the most likely user act is used. By using a systematic approach, the system is better able to handle conflicting evidence from different dialogue turns. Finally, the learned policies give significant improvements over the hand-crafted policies indicating that the learning has been effective.

4.4. User trial

Various authors have commented that evaluations using simulated users are not necessarily good predictors of performance with real human users. The various systems were therefore evaluated by 36 native English speakers, none of whom had been involved in previous experiments with dialogue systems. Users were asked to find a venue according to a set of constraints and then obtain some extra pieces of information. About 48 different task scenarios were used, including cases where there was no suitable venue, cases where there was more than one venue and cases where there was exactly one matching venue. For each system, every user was given one task from each of these three groups, resulting in a total of 108 dialogues recorded for each system.

Table 1
Objective success rate (OSR), objective score (OS) and subjective success rate (SSR) for the different systems. Error values give one standard error of the mean. They are calculated by assuming that success rates and rewards follow binomial and Gaussian distributions, respectively.

System	OSR	OS	SSR
BUDS–HDC	**0.84** $\pm$ 0.04	**11.83** $\pm$ 0.94	0.84 $\pm$0.04
BUDS–TRA	0.75 $\pm$ 0.04	8.89 $\pm$ 1.09	**0.88** $\pm$0.03
MDP–HDC	0.65 $\pm$ 0.05	7.10 $\pm$ 1.21	0.78 $\pm$ 0.04
MDP–TRA	0.66 $\pm$ 0.05	6.97 $\pm$ 1.23	0.81 $\pm$ 0.04

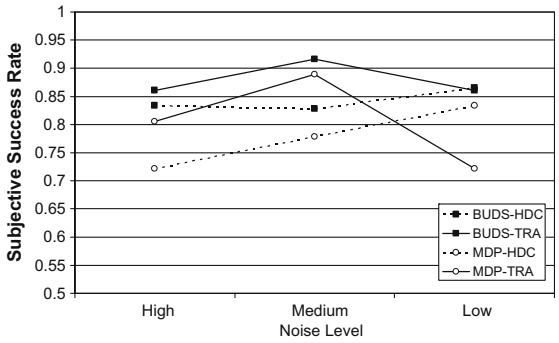

Fig. 10. Subjective success rates for the different systems at varying noise levels.

The robustness of the systems was evaluated by adding synthetic noise to the speech signal before speech recognition. Three levels of noise were used: low, medium and high corresponding to signal to noise ratios (SNR) of 35.3 db, 10.2 db and 3.3 db, respectively. Each user was tested under all three noise conditions for each system.

All four systems use the same speech recogniser, semantic decoder, output generator and text-to-speech engine. Speech recognition is implemented using the ATK toolkit with a tri-phone acoustic model and a dictionary of around 2000 in-domain words. The output of the recogniser is a 10-best list along with sentence-level inference evidence scores, which are the sum of the sentence arc log-likelihoods in the confusion network. A hand-crafted Phoenix-based parser is used for semantic decoding. The confidence for each resulting dialogue act is calculated by exponentiating the inference evidence, adding the score for sentences resulting in the same dialogue act and renormalising so that the sum is one. Simple template rules are used for output generation and text-to-speech uses a diphone synthesis engine.

Overall results from the trial are given in Table 1. It is clear from the results that the systems which use a statistical approach to modeling uncertainty give a significant improvement in performance. Interestingly, the BUDS trained policy outperformed the hand-crafted policy in terms of subjective success but was worse on the objective metrics. An investigation of the transcripts showed that the trained policy sometimes offered venues before all the information had been given. While the simulated user would always give further information to ensure its constraints were met, the human users did not necessarily do this. In these cases, the human user often thought the dialogue was successful when in fact some constraint was not met. This may be one reason for the reduced performance of the trained policy on the objective measures.

An analysis of the systems' robustness to noise was evaluated by separating out the results for different noise levels,[10] as shown in Fig. 10. The overall trend is that the BUDS systems do outperform the traditional approaches but these differences are not significant due to the small number of samples for each noise level. The graph shows a large variability in success rates which makes the results difficult to interpret. The objective success rates gave similar trends, although the BUDS–HDC policy performed better than the BUDS–TRA policy.

5. Conclusion

This paper has introduced a new POMDP-based framework for building spoken dialogue systems by using Bayesian updates of the dialogue state (BUDS). The BUDS model gives an efficient, statistically motivated approach to modeling the internal system state. Various modifications of standard Bayesian network algorithms were presented which give sufficient efficiency improvements over the standard loopy belief

[10] The questions asked by the dialogue manager can influence both the Word Error Rate (WER) as well as the Semantic Error Rate (SER). It is therefore more important to compare success under equivalent noise conditions than to compare success under equivalent error rates. Further experiments, not included in this paper, showed similar trends when the success rate was plotted as a function of the error rate instead of the noise level.

propagation algorithm to enable a real-world dialogue system to operate efficiently in real-time. The belief updating equations were also shown to be more efficient than alternative updating algorithms used in POMDP-based dialogue systems.

Dialogue systems using the BUDS framework can use either learned or hand-crafted policies. A learning algorithm based on the Natural Actor Critic algorithm was presented as a feasible approach to policy optimisation. The factorisation of the state space inherent in the BUDS model leads to a simple separation into components of the policy parameters. Domain knowledge may be incorporated by using summary actions to group together different actions and increase the tractability of the policy learning.

The BUDS framework was evaluated in both simulations and in a user trial. The two systems using the BUDS approach outperformed both the MDP and finite state systems. This difference was significant in both simulations and the user trial. The effect of policy learning with BUDS was not conclusive. In simulations the learned policy clearly outperforms the hand-crafted one but this did not translate into a conclusive improvement in the objective performance with human users. Users did, however, *perceive* the learned policy to be more successful than the hand-crafted policy.

One possible reason for the lower performance of the learned policy is the difference between the training and testing environments. The simulator, error channel and reward are all different between learning with the simulator and testing with human users. The policy is optimised for the reward which is dependent on the user's opinion of whether the dialogue was successful. This is arguably closer to the subjective success rather than to the objective success. Future learned policies should be able to conclusively outperform hand-crafted policies by using more appropriate rewards, user simulations, summary actions and basis function features.

Another reason for the reduced performance with human users is the choice of model parameters. The parameters of the model used here were estimated very roughly by inspecting their effects on dialogue evolution with the user simulator. It would clearly be preferable to learn these parameters from data by extending the Bayesian network to include the parameters. Unfortunately, Loopy Belief Propagation, would no longer be applicable because the parameter nodes are continuous and so a more general algorithm, such as Expectation Propagation, would be required. The authors are currently pursuing this topic.

It is clear from the paper that the use of a statistical approach to uncertainty can yield significant improvements in performance. Policy learning can also achieve competitive policies with little human intervention. The BUDS framework provides an approach that is efficient enough to scale to real-world problems, and which can outperform standard alternatives. Further work will attempt to learn the model parameters from data and to deploy the system in a live telephone application, showing that the benefits discussed here do translate into real-world improvements.

Acknowledgements

This research was funded by a St John's Benefactors Scholarship, the UK EPSRC under grant agreement EP/F013930/1 and by the EU FP7 Programme under grant agreement 216594 (CLASSIC project: www.classic-project.org). The authors would like to thank Jost Schatzmann, who built the user simulator and MDP based systems used here, as well as Milica Gašić, Filip Jurčíček, Simon Keizer, Fabrice Lefèvre, François Mairesse, Ulrich Paquet and Kai Yu for their help in the user trials and for useful comments and discussions.

Appendix A. General case of grouped loopy belief propagation

The grouped version of loopy belief propagation can be extended to any factor graph with discrete nodes. Consider the update for a factor, f_a, connected to a set of neighbouring discrete random variables, $X_1, X_2, \ldots, X_{\mathfrak{N}} \in ne(f_a)$. The corresponding portion of the factor graph is shown graphically in Fig. A.1.

Now suppose a partitioning of the space of values for each X_i, where $i = 1 \ldots \mathfrak{N}$. The partition for X_i is denoted here by the sets $Z_{i,1}, Z_{i,2}, \ldots, Z_{i,N_i}$, where $\sum_j P(X_i \in Z_{i,j}) = 1$. If $x \in Z_{i,j_1}$ and $x \in Z_{i,j_2}$ then $j_1 = j_2$.

The update equations for LBP require a summation over all combinations of X_i values. This will be grouped according to the partitions. To facilitate this the set of all combinations of partitions is now defined, along with the set of X_i values for a given partition.

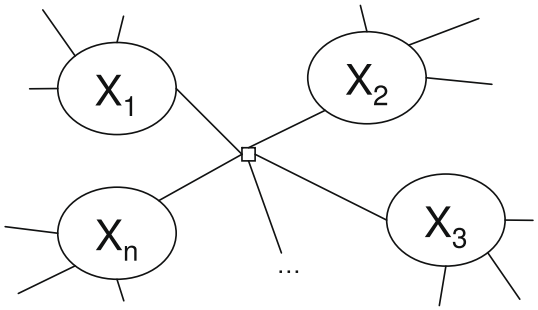

Fig. A.1. A portion of a factor graph.

Definition 1. The set of all combinations of partitions is a follows:

$$C = \{\mathbf{k} = (k_1, k_2, \ldots, k_{\mathfrak{N}}) : (\forall i) \quad k_i \in \mathbb{Z}, 0 < k_i \leqslant N_i\} \tag{A.1}$$

Definition 2. The set of all $\mathbf{x}$ vectors for a given partition is denoted $Y_{\mathbf{k}}$ and is given by the following equation:

$$Y_{\mathbf{k}} = \{\mathbf{x} = (x_1, x_2, \ldots, x_{\mathfrak{N}}) : (\forall i) \quad x_i \in Z_{i,k_i}\} \tag{A.2}$$

The requirements for concept values being indistinguishable are defined as follows:

Definition 3. The messages from the factor to variable X_i are said to be *constant with respect to the partition*, $(Z_{i,1}, \ldots, Z_{i,N_i})$ if:

$$(\forall j)(\forall x_i', x_i'' \in Z_{i,j}) \quad \mu_{X_i \to f_a}(x_i') = \mu_{X_i \to f_a}(x_i'') \tag{A.3}$$

In this case, the group input message is denoted by:

$$\mu_{X_i \to f_a}(Z_{i,j}) = \mu_{X_i \to f_a}(x_i) \text{ for some } x_i \in Z_{i,j} \tag{A.4}$$

The definition of *constant messages from the factor* is similar, with

$$\mu_{f_a \to X_i}(Z_{i,j}) = \mu_{f_a \to X_i}(x_i) \text{ for some } x_i \in Z_{i,j} \tag{A.5}$$

Definition 4. The set of partitions is said have *constant factor sums* with respect to a set of partitions, if for all neighbouring variables X_i

$$(\forall \mathbf{k} \in C)(\forall x_i', x_i' \in Z_{i,k_i}) \qquad \sum_{\mathbf{x} \in Y_{\mathbf{k}}, x_i = x_i'} f_a(\mathbf{x}) = \sum_{\mathbf{x} \in Y_{\mathbf{k}}, x_i = x_i'} f_a(\mathbf{x}) \tag{A.6}$$

Definition 5. Let the partition factor value, $f_a(\mathbf{k})$, be defined as follows:

$$f_a(\mathbf{k}) = f_a(Z_{1,k_1}, Z_{2,k_2}, \ldots Z_{n,k_{\mathfrak{N}}}) = \sum_{\mathbf{x} \in Y_{\mathbf{k}}} f_a(\mathbf{x}) \tag{A.7}$$

It is relatively intuitive that the message values should be constant across the values of a partition. The definition of constant factor sums is less obvious but turns out to be true in many cases. One simple example was given in Section 2.4.1. Assuming constant factor sums allows one to prove that the messages will stay constant for values in a partition.

Lemma 6. *If a set of partitions has constant factor sums over i and $x_i' \in Z_{i,j}$ for some j then*

$$f_a(\mathbf{k}) = |Z_{i,j}| \sum_{\mathbf{x} \in Y_{\mathbf{k}}, x_i = x_i'} f_a(\mathbf{x}) \tag{A.8}$$

Proof

$$f_a(\mathbf{k}) = \sum_{\mathbf{x} \in Y_{\mathbf{k}}} f_a(\mathbf{x}) = \sum_{x_i'' \in Z_{i,j}} \sum_{\mathbf{x} \in Y_{\mathbf{k}}, x_i = x_i''} f_a(\mathbf{x}) \tag{A.9}$$

$$= \sum_{x_i'' \in Z_{i,j}} \sum_{\mathbf{x} \in Y_{\mathbf{k}}, x_i = x_i'} f_a(\mathbf{x}) = |Z_{i,j}| \sum_{\mathbf{x} \in Y_{\mathbf{k}}, x_i = x_i'} f_a(\mathbf{x}) \quad \square \tag{A.10}$$

Proposition 7. *Suppose that for every variable X_i, the messages from the factor are constant with respect to the partition $Z_{i,1}, Z_{i,2}, \ldots, Z_{i,N_i}$. Suppose further that the set of partitions has constant factor sums. Then after updating messages from the factor to variable X_i, using Eq. (2) of LBP, all messages from the factor will still be constant with respect to the partition $Z_{i,1}, Z_{i,2}, \ldots, Z_{i,N_i}$.*

Proof. Suppose $x_i \in Z_{i,j}$ for some i, j. According to Eq. (2):

$$\mu_{f_a \to X_i}(x_i') = \sum_{\mathbf{x} \in Y, x_i = x_i'} f_a(\mathbf{x}) \prod_{l \neq i} \mu_{X_l \to f_a}(x_l) \tag{A.11}$$

$$= \sum_{\mathbf{k} \in C} \sum_{\mathbf{x} \in Y_{\mathbf{k}}, x_i = x_i'} f_a(\mathbf{x}) \prod_{l \neq i} \mu_{X_l \to f_a}(x_l) \tag{A.12}$$

$$= \sum_{\mathbf{k} \in C} \prod_{l \neq i} \mu_{X_l \to f_a}(Z_{l,k_l}) \sum_{\mathbf{x} \in Y_{\mathbf{k}}, x_i = x_i'} f_a(\mathbf{x}) \tag{A.13}$$

$$= \frac{1}{|Z_{i,j}|} \sum_{\mathbf{k} \in C} \prod_{l \neq i} \mu_{X_l \to f_a}(Z_{l,k_l}) f_a(\mathbf{k}) \tag{A.14}$$

Eq. (A.12) follows by enumerating out the partitions. Eq. (A.13) makes use of the constant message values within each partition, while (A.14) is a direct application of Lemma 6. $\square$

The above proposition shows that messages to the factor will be constant under the assumption of constant factor sums. The loopy belief propagation algorithm can update all the values for any partition simultaneously using Eq. (A.14). This is the same update formula as standard LBP, with the message values replaced by partition messages, the factor value replaced by the partition factor and an adjustment added for the number of values in the partition. Setting all partitions to singleton sets recovers standard LBP.

The input messages will also remain constant during updates as shown below:

Proposition 8. *Suppose that some variable X_i is connected to factors $f_b \in ne(X_i)$. Suppose further that the messages from all neighbouring factors are constant with respect to some partition $Z_{i,1}, X_{i,2}, \ldots, Z_{i,N_i}$. Then after updating messages to nearby factors according to Eq. (3) of LBP, all messages from X_i to connecting factors will still be constant with respect to the partition $Z_{i,1}, X_{i,2}, \ldots, Z_{i,N_i}$.*

Proof. Let $x_i' \in Z_{i,j}$ for some i, j. Then

$$\mu_{X_i \to f_b}(x_i') = \prod_{c \neq b} \mu_{f_c \to X_i}(x_i') = \prod_{c \neq b} \mu_{f_c \to X_i}(Z_{i,j}) \quad \square \tag{A.15}$$

These two propositions result in the essential property of this grouped form of LBP. Starting with constant messages and a set of partitions which have constant factor sums, loopy belief propagation will keep the messages constant with respect to the partitioning. This allows for significant efficiency gains because only one representative from each partition of concept values needs to be stored. At the same time, none of the efficiency gains due to LBP are lost.

Appendix B. Experimental analysis of the updating algorithms

B.1. Experimental models

In order to do an experimental analysis of the effect of the number of time-slices in the loopy belief propagation algorithm, a set of simple Bayesian networks were built. These give an abstraction of the type of structures that are present in dialogue systems (discussed in Section 2.2.1). Each time-slice of the network consists of a tree of goal nodes with depth D and branching factor B, along with an observation node for each goal node. Each node may take $B+1$ different values, labeled "N/A", $1, 2, \ldots, B$.

The observation nodes each depend on their associated goal node with observation probability given by:

$$p(o|g) = \begin{cases} \frac{5}{B+5} & \text{if } o = g \\ \frac{1}{B+5} & \text{otherwise} \end{cases}. \tag{B.1}$$

Each goal node has an associated parent goal, denoted g_p. Each value of the parent goal is associated with exactly one child node. The child goal will only take the values $1, \ldots, B$ in cases when the parent goal has this value, labeled a. In all other cases the child goal's value will be "N/A". The probability of moving to a particular goal g' given the previous goal g is then:

$$p(g'|g, g_p \neq a) = \begin{cases} 1 & \text{if } g' = N/A \\ 0 & \text{otherwise} \end{cases}, \tag{B.2}$$

$$p(g'|g, g_p = a) = \begin{cases} 0 & \text{if } g = N/A \\ \frac{5}{B+4} & \text{if } g' = g \neq N/A \\ \frac{1}{B+4} & \text{otherwise} \end{cases}. \tag{B.3}$$

In the first turn, a simple uniform distribution is used instead of the last two probabilities above – i.e. $p(g'|g, g_p = a) = \frac{1}{B}$ for $g' = g \neq N/A$.

This abstraction allows several tests of how the different algorithms compare in accuracy and speed for tasks of various complexity. Computation times using a depth of 0 and 2 were given in Figs. 5 and 6 of Section 2.6. Accuracies on a model with depth 1 and branching factor 5 are given below in Fig. B.1.

B.2. Accuracy analysis

An important feature of any approximate algorithm used for updating the belief distributions is its accuracy. In particular, the calculated distributions should closely match the true marginal distribution. One can

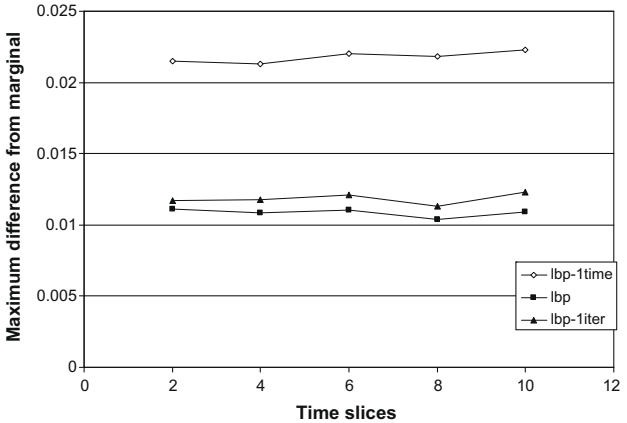

Fig. B.1. Experimental evaluation of the accuracy of various algorithms. All results use a hierarchical network with branching factor 5 and depth 1 (described in detail above). The graph shows the maximum difference for any probability value between the calculated marginals in the last time-slice and the true marginals as calculated by the junction tree algorithm. About 500 samples are taken from each network and the means are plotted. The standard errors of all estimates are below 0.001.

prove that the distributions calculated when maintaining only one time-slice will satisfy certain error bounds (Boyen and Koller, 1998). The accuracy of the general LBP algorithm is slightly more complex.

For tree structures, LBP is known to give an exact answer but in more general graphs, the fixed points of LBP may give slight variations from the true marginal distribution. One can show that LBP is a special case of expectation propagation applied to discrete variables (Minka, 2001) and as such, fixed points of the algorithm minimise a set of local KL divergences. Yedidia et al. (2001) and Heskes (2003) have shown that the stable fixed points also represent minima of the Bethe free energy function. In view of these results, one can expect LBP to give good approximations to the true marginal distributions.

Fig. B.1 presents a comparison of the accuracies of various algorithms on a tree network of depth 1 and branching factor 5 (details are given above). Three algorithms are presented:

- The LBP algorithm with one time-slice (lbp-1time). This is equivalent to the Boyen–Koller algorithm.
- The LBP algorithm with all available time-slices, iterating once backwards and forwards (lbp-1iter),
- Loopy belief propagation with all available time-slices, iterating until convergence (lbp).

Fig. B.1 shows how using the full history improves the accuracy significantly.[11] The single iteration version of LBP gives more accurate results than maintaining only one time-slice and a second iteration of LBP gives the convergent values which are again more accurate. Of course, the computation times using the full history are significantly longer and this will often be a more important factor than the slight drop in accuracy.

The results presented above are consistent with previous findings, which suggest that more history generally improves the accuracy (Murphy, 2002). It is interesting to note that there is little gain after the second iteration through the variables. On this problem it appears that LBP using the full history converges after only a couple of iterations.

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.

Boyen, X., Koller, D., 1998. Tractable inference for complex stochastic processes. In: Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence (UAI). Morgan Kaufmann, San Francisco, pp. 33–42.

Bui, T., Poel, M., Nijholt, A., Zwiers, J., 2007. A tractable DDN-POMDP approach to affective dialogue modeling for general probabilistic frame-based dialogue systems. In: Traum, D., Alexandersson, J., Jonsson, A., I.Z. (Eds.), Workshop on Knowledge and Reasoning in Practical Dialog Systems, International Joint Conference on Artificial Intelligence (IJCAI). Hyderabad, India, pp. 34–37.

Heskes, T., 2003. Stable fixed points of loopy belief propagation are minima of the Bethe free energy. In: Becker, S., Thrun, S., Obermayer, K. (Eds.), Advances in Neural Information Processing Systems, vol. 15. MIT Press, Cambridge, pp. 359–366.

Horvitz, E., Paek, T., 1999. A computational architecture for conversation. In: Proceedings of the Seventh International Conference on User Modeling, Wien Banff, Canada. Springer, New York, pp. 201–210.

Kaelbling, L.P., Littman, M.L., Cassandra, A.R., 1998. Planning and acting in partially observable stochastic domains. Artificial Intelligence 101, 99–134.

Kschischang, F., Frey, B., Loeliger, H., 2001. Factor graphs and the sum–product algorithm. IEEE Transactions on Information Theory 47, 498–519.

Lemon, O., Georgila, K., Henderson, J., Stuttle, M., 2006. An ISU dialogue system exhibiting reinforcement learning of dialogue policies: generic slot-filling in the TALK in-car system. In: Proceedings of the European chapter of the Association for Computational Linguistics (EACL).

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

Meng, H., Wai, C., Pieraccini, R., 2003. The use of belief networks for mixed-initiative dialog modeling. IEEE Transactions on Speech and Audio Processing 11 (6), 757–773.

Minka, T., 2001. A family of algorithms for approximate bayesian inference. Ph.D. thesis, MIT.

Murphy, K., 2002. Dynamic bayesian networks: Representation, inference and learning. Ph.D. thesis, UC Berkeley, Computer Science Division.

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

[11] An experimental evaluation using KL divergence gave the same trends.

Pieraccini, R., Huerta, J.M., 2008. Where do we go from here? In: Dybkjr, L., Minker, W. (Eds.), Recent Trends in Discourse and Dialogue, Text, Speech and Language Technology, vol. 39. Springer.

Pietquin, O., 2004. A Framework for Unsupervised Learning of Dialogue Strategies. SIMILAR Collection. Presses Universitaires de Louvain.

Pulman, S., 1996. Conversational games, belief revision and bayesian networks. In: Proceedings of the 7th Computational Linguistics in the Netherlands meeting.

Roy, N., Pineau, J., Thrun, S., 2000. Spoken Dialogue Management Using Probabilistic Reasoning. In: Proceedings of the Association for Computational Linguistics (ACL).

Schatzmann, J., 2008. Statistical user modeling for dialogue systems. Ph.D. thesis, University of Cambridge.

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

Scheffler, K., 2002. Automatic design of spoken dialogue systems. Ph.D. thesis, University of Cambridge.

Shani, G., Poupart, P., Brafman, R., Shimony, S., 2008. Efficient add operations for point-based algorithms. In: The International Conference on Automated Planning and Scheduling (ICAPS).

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

Sutton, R., McAllester, D., Singh, S., Mansour, Y., 2000. Policy gradient methods for reinforcement learning with function approximationAdvances in Neural Information Processing Systems, vol. 12. MIT Press, pp. 1057–1063.

Thomson, B., Schatzmann, J., Weilhammer, K., Ye, H., Young, S., 2007. Training a real-world POMDP-based dialog system. In: Proceedings of the HLT/NAACL workshop on "Bridging the Gap: Academic and Industrial Research in Dialog Technologies". Rochester, NY.

Thomson, B., Yu, K., Gasic, M., Keizer, S., Mairesse, F., Schatzmann, J., Young, S., 2008. Evaluating semantic-level confidence scores with multiple hypotheses. In: Proceedings of the Interspeech. Brisbane, Australia.

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

Williams, J.D., Young, S., 2006 a. Scaling POMDPs for dialog management with composite summary point-based value iteration (CSPBVI). In: Proceedings of the AAAI Workshop on Statistical and Empirical Approaches for Spoken Dialogue Systems. Boston.

Williams, J.D., 2007a. Applying POMDPs to dialog systems in the troubleshooting domain. In: Proceedings of the HLT/NAACL workshop on "Bridging the Gap:. Academic and Industrial Research in Dialog Technology, Rochester, NY, USA.

Williams, J.D., 2007b. Using particle filters to track dialogue state. In: IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU). Kyoto, Japan.

Williams, J.D., Poupart, P., Young, S., 2005. Factored partially observable Markov decision processes for dialogue management. In: Proceedings of the IJCAI Workshop on Knowledge and Reasoning in Practical Dialog Systems. Edinburgh.

Williams, J.D., Young, S., November 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., Young, S., 2006b. Partially observable markov decision processes for spoken dialog systems. Computer Speech and Language 21 (2), 231–422.

Williams, J.D., Young, S., 2007. Scaling POMDPs for spoken dialog management. IEEE Transactions on Audio, Speech, and Language Processing 15 (7), 2116–2129.

Yedidia, J.S., Freeman, W.T., Weiss, Y., 2001. Generalized belief propagation. Advances in Neural Information Processing Systems, vol. 13. MIT Press, pp. 689–695.

Young, S., Schatzmann, J., Weilhammer, K., Ye, H., 2007. The Hidden Information State Approach to Dialog Management. In: Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP). Honolulu, Hawaii.