Learning What to Talk About in Descriptive Games

Hugo Zaragoza
Microsoft Research
Cambridge, United Kingdom
hugoz@microsoft.com

Chi-Ho Li
University of Sussex
Brighton, United Kingdom
C.H.Li@sussex.ac.uk

Abstract

Text generation requires a planning module to select an object of discourse and its properties. This is specially hard in *descriptive games*, where a computer agent tries to describe some aspects of a game world. We propose to formalize this problem as a Markov Decision Process, in which an optimal message policy can be defined and learned through simulation. Furthermore, we propose *back-off policies* as a novel and effective technique to fight state dimensionality explosion in this framework.

1 Introduction

Traditionally, text generation systems are decomposed into three modules: the *application* module which manages the high-level task representation (state information, actions, goals, etc.), the *text planning* module which chooses messages based on the state of the application module, and the *sentence generation* module which transforms messages into sentences. The planning module greatly depends on the characteristics of both the application and the generation modules, solving issues in domain modelling, discourse and sentence planning, and to some degree lexical and feature selection (Cole et al., 1997). In this paper we concentrate on one of the most basic tasks that text planning needs to solve: selecting the message content, or more simply, choosing *what to talk about*.

Work on text-generation often assumes that an object or topic has been already chosen for discussion. This is reasonable for many applications, but in some cases choosing *what* to talk about can be harder than choosing *how* to. This is the case in the type of text generation applications that we are interested in: generating descriptive messages in computer games. In a modern computer game at any given moment there may be an enormous number of object properties that can be described, each with varying importance and consequences. The outcome of the game depends not only on the skill of the player, but also on the quality of the descriptive messages produced. We refer to such situations as *descriptive games*.

Our goal is to develop a strategy to choose the most interesting descriptive messages that a particular talker may communicate to a particular listener, given their context (i.e. their knowledge of the world and of each-other). We refer to this as *message planning*.

Developing a general framework for planning is very difficult because of the strong coupling between the planning and application modules. We propose to frame message planning as a Markov Decision Process (MDP) which encodes the environment, the information available to the *talker* and *listener*, the consequences of their communicative and non-communicative acts, and the constraints of the text generation module. Furthermore we propose to use Reinforcement Learning (RL) to learn the optimal message policy. We demonstrate the overall principle (Section 2) and then develop in more detail a computer game setting (Section 3).

Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing (HLT/EMNLP), pages 291–298, Vancouver, October 2005.

One of the main weaknesses of RL is the problem of state dimensionality explosion. This problem is specially acute in message planning, since in typical situations there can be hundreds of thousands of potential messages. At the same time, the domain is highly structured. We propose to exploit this structure using a form of the back-off smoothing principle on the state space (Section 4).

1.1 Related Work

Our problem setting can be seen as a generalisation of the *content selection* problem in the generation of referring expressions in NLG. In the standard setting of this problem (see for example (van Deemter and Krahmer, to appear)) an algorithm needs to select the *distinguishing description* of an object in a scene. This description can be seen as a subset of scene properties which i) uniquely identifies a given target object, and ii) is optimal in some sense (minimal, psychologically plausible, etc.) van Deemter and Krahmer show that most content selection algorithms can be described as different cost functions over a particular graph representation of the scene. Minimising the cost of a subgraph leads to a distinguishing description.

Some aspects of our work generalise that of content selection: i) we consider the target object is unknown, ii) we consider scenes (i.e. world states) that are *dynamic* (i.e. they change over time) and *reactive* (i.e. utterances change the world), and iii) we consider listeners that have partial knowledge of the scene. This has important consequences. For example, the cost of a description cannot be directly evaluated; instead, we must *play the game*, that is, generate utterances and observe the rewards obtained over time. Also identical word-states may lead to different optimal messages, depending on the listener's partial knowledge. Other aspects of our work are very simplistic compared to current work in content selection, for example with respect to the use of negation and of properties that are boolean, relative or graded (van Deemter and Krahmer, to appear). We hope to incorporate these ideas into our work soon.

Probabilistic dialogue policies have been previously proposed for spoken dialogue systems (SDS) (see for example (Singh et al., 2002; Williams et al., 2005) and references therein). However, work in SDS focus mainly on coping with the noise and uncertainty resulting from speech recognition and sentence parsing. In this context MDPs are used to infer features and plan communicative strategies (modality, confusion, initiative, etc.) In our work we do not need to deal with uncertainty or parsing; our main concern is in the selection of the message content. In this sense our work is closer to (Henderson et al., 2005), where RL is used to train a SDS with very many states encoding message content.

Finally, with respect to the state-explosion problem in RL, related work can be found in the areas of multi-task learning and robot motion planning (Dietterich, 2000, and references therein). In these works the main concern is identifying the features that are relevant to specific sub-tasks, so that robots may learn multiple loosely-coupled tasks without incurring state-explosion. (Henderson et al., 2005) also addresses this problem in the context of SDS and proposes a semi-supervised solution. Our approach is related to these works, but it is different in that we assume that the feature structure is known in advance and has a very particular form amenable to a form of back-off regularisation.

2 Message planning

Let us consider an environment comprising a world with some objects and some agents, and some dynamics that govern their interaction. Agents can observe and memorize certain things about the world, can carry out actions and communicate with other agents. As they do so, they are rewarded or punished by the environment (e.g. if they find food, if the complete some goal, if they run out of energy, etc.)

The agents' actions are governed by a *policy*. We will consider separately the *physical action policy* (π), which decides which physical action to take given the state of the agent, and the *message action policy* (μ), which decides when to communicate, to whom, and what about. Our main concern in this paper will be to learn an optimal μ. Before we define this goal more precisely, we will introduce some notation.

A *property* is a set of *attribute-value pairs*. An *object* is a set of properties, with (at least) attributes Type and Location. A *domain* is a set of objects. Fur-

thermore, we say that s' is a *sub-domain* of s if s' can be obtained by deleting property–value pairs from s (while enforcing the condition that remaining objects must have Type and Location). $\mathsf{Sub}(s)$ is the set containing s, all sub-domains of s, and the empty domain $\emptyset$.

A *world state* can be represented as a domain, noted s_W. Any partial view of the world state can also be represented as a domain $s \in \mathsf{Sub}(s_W)$. Similarly the content of any descriptive message about the world, noted m, can be represented as a partial view of it. An *agent* is the tuple:

$$A := \left(s_A, \pi_A, \{\mu_{AA'}, s_{AA'}\}_{A' \neq A}\right)$$

- $s_A \in \mathsf{Sub}(s_W)$: knowledge that A has about the state of the world.
- $s_{AA'} \in \mathsf{Sub}(s_A \cap s'_A)$: knowledge that A has about the knowledge that A' has about the world.
- $\pi_a := P(c|s_A)$ is the action policy of A, and c is a physical action.
- $\mu_{AA'} := P(m \in \mathcal{M}(s_A)|s_A, s_{AA'})$ is the message policy of A for sending messages to A', and $\mathcal{M}(s_A)$ are all valid messages at state s_A (discussed in Section 2.3).

When an agent A decides to send a message to A', it can use its knowledge of A' to choose messages effectively. For example, A will prefer to describe things that it knows A' does not know (i.e. not in $s_{AA'}$). This is the reason why the message policy μ_A depends on both s_A and $s_{AA'}$. After a message is sent (i.e. realised and uttered) the agent's will update their knowledge states $s_{A'}$, $s_{A'A}$ and $s_{AA'}$.

The question that we address in this paper is that of *learning an optimal message policy* $\mu_{AA'}$.

2.1 Talker's Markov Decision Process

We are going to formalize this problem as a standard Markov Decision Process (MDP). In general a MDP (Sutton and Barto, 1998) is defined over some set of states $\mathcal{S} := \{s_i\}_{i=1..K}$ and actions associated to every state, $\mathcal{A}(s_i) := \{a_{ij}\}_{j=1..N_i}$. The environment is governed by the state transition function $\mathcal{P}^a_{ss'} := P(s'|s, a)$. A policy determines the likelihood of actions at a given state: $\pi(s) := P(a|s)$. At each state transition a reward is generated from the reward function $\mathcal{R}^a_{ss'} := E\{r|s, s', a\}$.

MDPs allow us to define and find *optimal policies* which maximise the expected reward. Classical MDPs assume that the different functions introduced above are known and have some tractable analytical form. Reinforcement Learning (RL) in as extension of MDPs in which the environment function $\mathcal{P}^a_{ss'}$ is unknown or complex, and so the optimal policy needs to be learned online by directly interacting with the environment. There exist a number of algorithms to solve a RL problem, such as Q-Learning or SARSA (Sutton and Barto, 1998).

We can use a MDP to describe a full descriptive game, in which several agents interact with the world and communicate with each-other. To do so we would need to consider composite states containing s_W, $\{s_A\}_A$, and $\left\{\{s_{AA'}\}_{A' \neq A}\right\}_A$. Similarly, we need to consider composite policies containing $\{\pi_A\}_A$ and $\left\{(\mu_{AA'})_{A' \neq A}\right\}_A$. Finally, we would consider the many constrains in this model; for example: only physical actions affect the state of the world, only message actions affect believes, and only believe states can affect the choice of the agent's actions.

MDPs provide us with a principled way to deal with these elements and their relationships. However, dealing with the most general case results in models that are very cumbersome and which hide the conceptual simplicity of our approach. For this reason, we will limit ourselves in this paper to one of the simplest communication cases of interest: a single all-knowing talker, and a single listener completely observed by the talker. We will discuss later how this can be generalized.

2.2 The *Talking God* Setting

In the simplest case, an all-knowing agent A_0 sits in the background, without taking any physical actions, and uses its message policy (μ_{01}) to send messages to a *listener* agent A_1. The listener agent cannot talk back, but can interact with the environment using its physical action policy π_1. Rewards obtained by A_1 are shared by both agents. We refer to this setting as the *talking God* setting. Examples of such situations are common in games, for example when a computer character talks to its (computer) team-

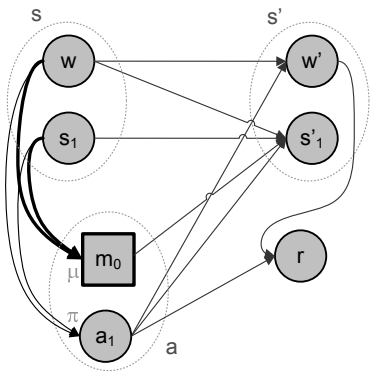

Figure 1: *Talking God* MDP.

mates, or when a mother-ship with full information of the ground sends a small simple robot to do a task. Another example would be that of a teacher talking to a learner, except that the teacher may not have full information of the learners head!

Since the talker is all-knowing, it follows that $s_0 = s_W$ and $s_{01} = s_1$. Furthermore, since the talker does not take physical actions, π_0 does not need to be defined. Similarly, since the listener does not talk we do not need to define μ_{10} or s_{10}. This case is depicted in Figure 1 as a graphical model. By grouping states and actions (dotted lines) we can see that this is can be modelled as a standard MDP. If all the probability distributions are known analytically, or if they can be sampled, optimal physical and message policies can be learnt (thick arrows).

Several generalizations of this model are possible. A straight forward generalization is to consider more than one listener agent. We can then choose to learn a single policy for all, or individual policies for each agent.

A second way to generalize the setting is to make the listeners mind only partially observable to the talker. In this case the talker continues to know the entire world ($s_0 = s_W$), but does not know exactly what the listener knows ($s_{01} \neq s_0$). This is more realistic in situations in which the listener cannot *talk back* to the talker, or in which the talkers mind is not observable. However, to model this we need a partially observable MDP (POMDP). Solving POMDPS is much harder than solving MDPs, but there have been models proposed for dialogue management (Williams et al., 2005).

In the more general case, the talker would have partial knowledge of the world and of the listener, and would itself act. In that case all agents are equal and can communicate as they evolve in the environment. The other agents minds are not directly observable, but we obtain information about them from their actions and their messages. This can all be in principle modelled by POMDPs in a straightforward manner, although solving these models is more involved. We are currently working towards doing so.

Finally, we note that all the above cases have dealt with worlds in which objects are static (i.e. information does not become obsolete), agents do not gain or communicate erroneous information, and communication itself is non-ambiguous and lossless. This is a realistic scenario for text generation, and for communication between computer agents in games, but it is far removed from the spoken dialogue setting.

2.3 Generation Module and Valid Messages

Generating descriptive sentences of domains can be done in a number of ways, from template to feature-based systems (Cole et al., 1997). Our framework does not depend on a particular choice of generation module, and so we do not need to discuss this module. However, our message policy is not decoupled of the generation module; indeed, it would not make sense to develop a planning module which plans messages that cannot be realised! In our framework, the generation module is seen simply as a fixed and known *filter* over all possible the messages.

We formalize this by representing an agent's generation module as a function $\Gamma_A(m)$ mapping a message m to a NL sentence, or to $\emptyset$ if the module cannot fully realise m. The set of available messages to an agent A in state s_A is therefore: $\mathcal{M}(s_A) := \{m \,|\, m \in \mathsf{Sub}(s_A)\,,\ \Gamma_A(m) \neq \emptyset\}$.

3 A Simple Game Example

In this section we will use a simple computer game to demonstrate how the proposed framework can be used to learn message policies.

The game evolves in a grid-world. A mother-ship sends a *scout*, which will try to move from its

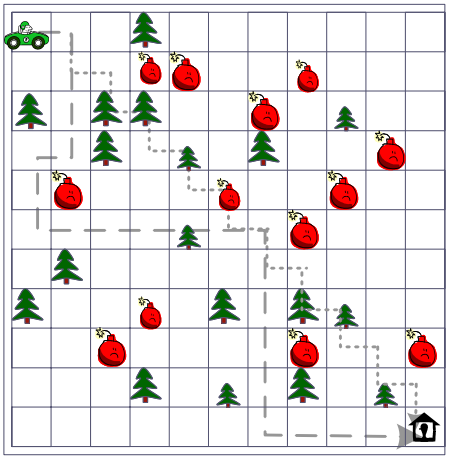

Figure 2: Example of a Simple Game Board.

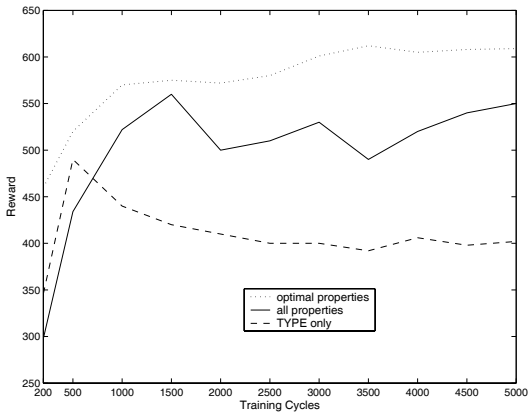

Figure 3: Simple Game Learning Results

State	**Best Action Learnt** *(and possible sentence realisation)*
{ TREE-BIG-LEFT }	$\emptyset$ *-SILENCE-*
{ BOMB-BIG-FRONT }	BOMB-FRONT *There is a bomb in front of you*
{ TREE-SMALL-LEFT, TREE-BIG-RIGHT }	TREE-BIG-RIGHT *There is a big tree to your right*
{ BOMB-BIG-FRONT, BOMB-SMALL-LEFT, TREE-BIG-RIGHT, TREE-SMALL-BACK }	TREE-BIG-RIGHT *There is a big tree to your right*

Table 1: Examples of learnt actions.

starting position (top left corner) to a target (bottom right). There are two types of objects on the board, Type $:= \{\mathsf{bomb}, \mathsf{tree}\}$, with a property Size $:= \{\mathsf{big}, \mathsf{small}\}$ in addition of Location. If a scout attempts to move into a big tree, the move is blocked; small trees have no effect. If a scout moves into a bomb the scout is destroyed and a new one is created at the starting position. Before every step the mother-ship may send a message to the scout. Then the scout moves one step (horizontal or vertical) towards the target choosing the shortest path which avoids hazards known by the scout (the A* algorithm is used for this). Initially scouts have no knowledge of the objects in the world; they gain this knowledge by stepping into objects or by receiving information from the mother-ship.

This is an instance of the talking god model discussed previously. The scout is the listener agent (A_1), and the mother-ship the talker (A_0). The scouts action policy π_1 is fixed (as described above), but we need to learn the message policy μ_{01}.

Rewards are associated with the results of physical actions: a high positive reward (1000) is assigned to reaching the destination, a large negative reward (-100) to stepping in a bomb, a medium negative reward (-10) to being blocked by a big tree, a small negative reward to every step (-1). Furthermore, sending a message has a small negative reward proportional to the number of attributes mentioned in the message (-2 per attribute, to discourage the talker from sending useless information). The message $\emptyset$ is given zero cost; this is done in order to learn when *not* to talk.

Learning is done as follows. We designed five maps of 11×11 cells, each with approximately 15 bombs and 20 trees of varying sizes placed in strategic locations to make the scouts task difficult (one of these maps is depicted in Figure 2; an A* path without any knowledge and one with full knowledge of the board are shown as dotted and dashed arrows respectively). A training epoch consists of randomly drawing one of these maps and running a single game until completion. The SARSA algorithm is used to learn the message policy, with $\epsilon = 0.1$ and $\gamma = 0.9$. The states s_W and s_1 are encoded to represent the location of objects surrounding the scout, relative to its direction (i.e. objects directly in front of the agent always receive the same location value). To speed up training, we only consider the 8 cells adjacent to the agent.

Figure 3 shows the results of these experiments. For comparison, we note that completing the game

with a uniformly random talking policy results in an average reward of less than -3000 meaning that on average more than 30 scouts die before the target is reached. The dashed line indicates the reward obtained during training for a policy which does not use the size attribute, but only type and location. This policy effectively learns that both bombs and trees in front of the agent are to be communicated, resulting in an average reward of approximately 400, and reducing the average number of deaths to less than 2. The solid line represents the results obtained by a policy that is forced to use all attributes. Despite the increase in communication cost, this policy can distinguish between small and large trees, and so it increases the overall reward two-fold. Finally, the dotted line represents the results obtained by a policy that can choose whether to use or not the size attribute. This policy proves to be even more effective than the previous one; this means that it has learnt to use the size attribute only when it is necessary. Some optimal (state,action) pairs learnt for this policy are shown in Table 1. The first three show correctly learnt optimal actions. The last is an example of a wrongly learnt action, due to the state being rare.

These are encouraging results, since they demonstrate in practice how optimal policies may be learnt for message planning. However, it should be clear form this example that, as we increase the number of types, attributes and values, this approach will become unfeasible. This is discussed in the next section.

4 Back-Off Policies

One of the main problems when using RL in practical settings (and, more generally, using MDPs) is the exponential growth of the state space, and consequently of the learning time required. In our case, if there are M attributes, and each attribute p_i has $N(p_i)$ values, then there are $S = \prod_{i=1}^{M} N(p_i)$ possible sub-domains, and up to 2^S states in the state space. This exponential growth, unless addressed, will render MDP learning unfeasible.

NL domains are usually rich with structure, some of it which is known *a priori*. This is the case in text generation of descriptions for computer games, where we have many sources of information about the objects of discourse (i.e. world ontology, dynamics, etc.) We propose to tackle the problem of state dimensionality explosion by using this structure explicitly in the design of hierarchical policies.

We do so by borrowing the back-off smoothing idea from language models. This idea can be stated as: train a set of probability models, ordered by their specificity, and make predictions using the most specific model possible, but only if there is enough training data to support its prediction; otherwise, *back-off* to the next less-specific model available.

Formally, let us assume that for every state s we can construct a sequence of K embedded partial representations of increasing complexity, $(s_{[1]}, \ldots, s_{[k]}, \ldots, s_{[K]})$. Let us denote $\hat{\pi}_{[k]}$ a sequence of policies operating at each of the partial representation levels respectively, and let each of these policies have a *confidence measurement* $c_k(s)$ indicating the quality of the prediction at each state. Since k indicates increasingly complex, we require that $c_k(s) \geq c_{k'}(s)$ if $k < k'$. Then, the most specific policy we can use at state s can be written as:

$$k_s^* := \arg\max_k \{k \cdot \operatorname{sign}(c_k(s) - \theta)\} \qquad (1)$$

A back-off policy can be implemented by choosing, at every state s the most specific policy available:

$$\pi(s) = \hat{\pi}_{[k_s^*]}(s_{[k_s^*]}) \qquad (2)$$

We can use a standard off-policy learning algorithm (such as Q-learning or SARSA) to learn all the policies simultaneously. At every step, we draw an action using (2) and update all policies with the obtained reward[1]. Initially, the learning will be driven by high-level (simple) policies. More complex policies will kick-in progressively for those states that are encountered more often.

In order to implement back-off policies for our setting, we need to define a confidence function c_k. A simple confidence measure is the number of times the state $s_{[k]}$ has been previously encountered. This measure grows on average very quickly for small k states and slowly for high k states. Nevertheless, reoccurring similar states will have high visit counts

[1] An alternative view of back-off policies is to consider that a single complete policy is being learnt, but that actions are being drawn from regularised versions of this policy, where the regularisation is a back-off model on the features. We show this in Appendix I

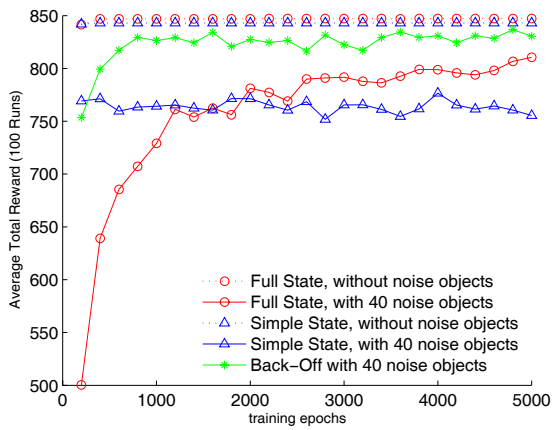

Figure 4: Back-Off Policy Simulation Results.

for all k values. This is exactly the kind of behaviour we require.

Furthermore, we need to choose a set of representations of increasing complexity. For example, in the case of n-gram models it is natural to choose as representations sequences of preceding words of increasing size. There are many choices open to us in our application domain. A natural choice is to order attribute types by their importance to the task. For example, at the simplest level of representation objects can be represented only by their type, at a second level by the type and colour, and at a third level by all the attributes. This same technique could be used to exploit ontologies and other sources of knowledge. Another way to create levels of representation of increasing detail is to consider different perceptual windows. For example, at the simplest level the agent can consider only objects directly in front of it, since these are generally the most important when navigating. At a second level we may consider also what is to the left and right of us, and finally consider all surrounding cells. This could be pursued even further by considering regions of increasing size.

4.1 Simulation Results

We present here a series of experiments based on the previous game setting, but further simplified to pinpoint the effect of dimensionality explosion, and how back-off policies can be used to mitigate it.

We modify the simple game of Section 3 as follows. First, we add a new object type, stone, and a new property Colour $:= \{$red, green$\}$. We let all trees be green and big and all bombs red and small, and furthermore we fix their location (i.e. we use one map instead of five). Finally we change the world behaviour so that an agent that steps into a bomb receives the negative reward but does not die, it continues until it reaches the target. All these changes are done to reduce the variability of our learning baseline.

At every game we generate 40 stones of random location, size and colour. Stepping on stones has no physical effect to the scout and it generates the same reward as moving into an empty cell, but this is unknown to the talker and will need to be learnt. These stones are used as *noise* objects, which increase the size of the state space. When there are no noise objects, the number of possible states is $3^8 \approx 6.5K$ (the actual number of states will be much smaller since there is a single maze). Noise objects can take $2 \times 2 = 4$ possible forms, so the total number of states with noise objects is $(3+4)^8 \approx 6M$. Even with such a simplistic example we can see how drastic the state dimensionality problem is. Despite the fact that the noise objects do not affect the reward structure of our simple game, reinforcement learning will be drastically slowed down by them.

Simulation results[2] are shown in Figure 4. First let us look at the results obtained using the full state representation used in Section 3 (noted Full State). Solid and dotted lines represent runs obtained with and without noise objects. First note that learning without noise objects (dotted circles) occurs mostly within the first few epochs and settles after 250 epochs. When noise objects are added (solid circles) learning greatly slows down, taking over 5K epochs. This is a typical illustration of the effect that the number of states has on the speed of learning.

An obvious way to limit the number of states is to eliminate features. For comparison, we learned a simple representation policy with states encoding only the type of the object *directly in front* of the agent, ignoring its colour and all other locations (noted Simple State). Without noise, the performance (dotted triangles) is only slightly worse than that of the original policy. However, when noise objects

[2]Every 200 training epochs we run 100 validation epochs with $\epsilon = 0$. Only the average validation rewards are plotted.

are added (solid triangles) the training is no longer slowed down. In fact, with noise objects this policy outperforms the original policy up to epoch 1000: the performance lost in the representation is made up by the speed of learning.

We set up a back-off policy with $K = 3$ as follows. We use the Simple representation at $k = 1$, plus a second level of representation where we represent the colour as well as the type of the object in front of the agent, and finally the Full representation as the third level. As the c_k function we use state visit counts as discussed above and we set $\theta = 10$. Before reaching the full policy (level 3), this policy should progressively learn to avoid bombs and trees directly in front (level 1), then (level 2) not avoid small trees directly in front. We plot the performance of this back-off policy (stars) in Figure 4. We see that it attains very quickly the performance of the simple policy (in less than 200 epochs), but the continues to increase in performance settling within 500 epochs with a performance superior to that of the full state representation, and very close to that of the policies operating in the noiseless world.

Despite the small scale of this study, our results clearly suggest that back-off policies can be used effectively to control state dimensionality explosion when we have strong prior knowledge of the structure of the state space. Furthermore (and this may be very important in real applications such as game development) we find that back-off policies produce a *natural* to feel to the errors incurred while learning, since policies develop progressively in their complexity.

5 Conclusion

We have developed a formalism to learn interactively the most informative message content given the state of the listener and the world. We formalised this problem as a MDP and shown how RL may be used to learn message policies even when the environment dynamics are unknown. Finally, we have shown the importance of tackling the problem of state dimensionality explosion, and we have proposed one method to do so which exploits explicit *a priori* ontological knowledge of the task.

References

R. Cole, J. Mariani, H. Uszkoreit, A. Zaenen, and V. Zue. 1997. *Survey of the State of the Art in Human Language Technology*. Cambridge University Press.

T. G. Dietterich. 2000. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303.

J. Henderson, O. Lemon, and K. Georgila. 2005. Hybrid reinforcement/supervised learning for dialogue policies from communicator data. In *4th IJCAI Workshop on Knowledge and Reasoning in Practical Dialogue Systems*.

S. Singh, D. Litmanand, M. Kearns, and M. Walker. 2002. Optimizing dialogue management with reinforcement learning: Experiments with the njfun system. *Journal of Artificial Intelligence Research*, 16:105–133.

R. S. Sutton and A. G. Barto. 1998. *Reinforcement Learning*. MIT Press.

K. van Deemter and E. Krahmer. (to appear). Graphs and booleans. In *Computing Meaning*, volume 3 of *Studies in Linguistics and Philosophy*. Kluwer Academic Publishers.

J. D. Williams, P. Poupart, and S. Young. 2005. Factored partially observable markov decision processes for dialogue management. In *4th IJCAI Workshop on Knowledge and Reasoning in Practical Dialogue Systems*.

6 Appendix I

We show here that the expected reward for a partial policy π_k after an action a, noted $Q^{\pi_k}(s,a)$, can be obtained from the expected reward of the full policy $Q^{\pi}(s,a)$ and the conditional state probabilities $P(s|s_{[k]})$. We may use this to compute the expected risk of any partial policy $R^{\pi_k}(s)$ from the full policy.

Let $\mathcal{T}_k(s) := \left\{ s' \in \mathcal{S} \mid s'_{[k]} = s_{[k]} \right\}$ be the subset of full states which map to the same value of s. Given a state distribution $P(s)$ we can define distributions over partial states:

$$P(s_{[k]}, s_{[j]}) = \sum_{s' \in \mathcal{T}_k(s) \cap \mathcal{T}_j(s)} P(s') \quad . \qquad (3)$$

Since $\sum_{s' \in \mathcal{T}_k(s)} P(s'|s_{[k]}) = 1$, we have $P(A|s_{[k]}) = \sum_{s' \in \mathcal{T}_k(s)} P(A|s') P(s'|s_{[k]})$, and so:

$$Q^{\pi_k}(s,a) = \sum_{s' \in \mathcal{T}_k(s)} P(s'|s_{[k]}) Q^{\pi}(s',a) \quad . \qquad (4)$$