Hierarchical Reinforcement Learning for Situated Natural Language Generation†

NINA DETHLEFS and HERIBERTO CUAYÁHUÍTL
Heriot-Watt University,
Mathematical and Computer Sciences,
Edinburgh, UK.

(Received 4 August 2014)

Abstract
Natural Language Generation systems in interactive settings often face a multitude of choices, given that the communicative effect of each utterance they generate depends crucially on the interplay between its physical circumstances, addressee and interaction history. This is particularly true in interactive and situated settings.

In this article, we present a novel approach for situated Natural Language Generation in dialogue that is based on hierarchical reinforcement learning and learns the best utterance for a context by optimisation through trial and error. The model is trained from human-human corpus data and learns particularly to balance the trade-off between efficiency and detail in giving instructions: the user needs to be given sufficient information to execute their task, but without exceeding their cognitive load.

We present results from simulation and a task-based human evaluation study comparing two different versions of hierarchical reinforcement learning: one operates using a hierarchy of policies with a large state space and local knowledge, and the other additionally shares knowledge across generation subtasks to enhance performance. Results show that sharing knowledge across subtasks achieves better performance than learning in isolation, leading to smoother and more successful interactions that are better perceived by human users.

1 Introduction
Natural Language Generation (NLG) systems across domains typically face an uncertainty with respect to the best utterance to generate in a given context. This is particularly true in interactive scenarios that involve constant verbal or non-verbal

† This research was supported by the German Research Foundation DFG under a grant for the Transregional Research Centre SFB/TR8 Spatial Cognition, project I5-DiaSpace. It was also supported by the European Commission’s FP7 programmes under grant agreement no. 287615 (PARLANCE) and no. ICT-248116 (ALIZ-E). We would like to thank John Bateman, Jette Viethen, Alexander Koller, Konstantina Garoufi, Kristina Striegitz, Oliver Lemon, Michael Strube and David Schlangen for comments and interesting discussions on the work presented. A special thanks to Kristina Striegitz and Konstantina Garoufi for helping us make sense of the GIVE challenge data.
feedback from a human user. The reason is that utterances can have different effects depending on the physical circumstances, addressee and interaction history of the context in which they occur. This article presents a hierarchical optimisation approach for situated NLG.

Situated NLG can be defined as generation in an enriched physical context, including features of a (real or virtual) environment, such as landmarks and users. The context in this setting is typically not static but undergoes dynamic changes triggered by linguistic or non-linguistic actions by the system or the user. Often, as in our case, situated NLG also deals with an additional element of interactivity in that the user can immediately react to the system’s instructions through linguistic or non-linguistic actions. Figure 1 shows an example of the type of generation scenario we will address in this article. It shows a spatial situation (from the perspective of the user) and a set of possible instructions which differ with respect to their level of granularity in identifying the (circled) referent button. A trade-off in situated NLG is often between generating efficient instructions and detailed instructions. Since the user is constantly moving through a virtual world, instructions need to contain just the right amount of information so that the user’s cognitive load remains low and they do not get lost. In the figure, only instruction (c) seems to balance this trade-off appropriately. Instruction (a) is ambiguous and instruction (b) is complete, but long and difficult to memorise for a user on the move.

While different techniques are conceivable to address this efficiency vs detail trade-off, we will present an optimisation framework that is based on hierarchical reinforcement learning and optimises its decision making over time through a trial and error search. To this end, we design a hierarchy of learning agents, each of them representing a specific generation subtask. A hierarchical policy is then trained from interaction with a simulated environment which was trained from a corpus of human-human interactions. We argue that using reinforcement learning, an NLG agent is able to try a multitude of generation strategies under different circumstances and discover the optimal one automatically.

The hierarchical setup offers the additional benefit of a divide-and-conquer approach. This provides a modular and easy-to-maintain architecture, makes learning
faster and our technique more scalable than flat reinforcement learning setups due to the reduced policy search space. A possible disadvantage of using a modular architecture is that knowledge variables are specific to a particular generation subtask, such as referring expression generation or navigation. This automatically assumes an independence among subtasks which may not necessarily hold in practice. We therefore compare two different versions of a hierarchical reinforcement learner in this article: one that shares task-based knowledge across generation subtasks, using a joint optimisation, and one that does not, using an isolated optimisation. Shared knowledge is pre-defined by the system designer. Our hypothesis is that by sharing knowledge the learning agent becomes more aware of the global effects of its actions rather than being confined to the local context of a particular subtask. By trying alternative sequences of decisions and observing the user’s reactions, the system then becomes able to predict their effects on the utterance as a whole.

The article is organised as follows. In Section 2, we review related work in three areas: (i) the application of reinforcement learning to NLG, (ii) the sharing of knowledge across subtasks, and (iii) the state-of-the-art in situated NLG. Section 3 will then introduce the GIVE task, the situated scenario we are addressing in this article. Subsequently, Section 4 will give an overview of flat and hierarchical reinforcement learning and discuss its application to situated NLG. We will present the learning agent’s training setting in Section 5, followed by an evaluation in Section 6. The evaluation consists of two parts: (a) a simulation-based evaluation; and (b) a task-based evaluation comparing joint and isolated policy learning for hierarchical reinforcement learning. It also makes a comparison with other state-of-the-art approaches to situated NLG. Finally, Section 7 will draw conclusions and discuss directions for future research.

2 Related work

In this section, we will review previous research on applying reinforcement learning to optimising sequences of NLG decisions and its relation to planning approaches. Further, we will discuss the sharing of knowledge across application subtasks and the state-of-the-art in situated NLG. For each strand of work, we highlight commonalities and differences with our proposed approach.

2.1 Reinforcement learning for NLG in interactive systems

Reinforcement learning (RL) has become a popular method for optimising dialogue management decisions for flat (Singh et al., 2002) and hierarchical decision problems (Cuayáhuithl et al., 2010). It has been appreciated especially for its ability of automatic optimisation, discovery of fine-grained behaviour from human data and adaptability under uncertain circumstances (Williams and Young, 2007).

The NLG community has successfully adopted RL rather recently and with a specific focus on optimising generation for interactive systems (Lemon, 2011). Rieser, Lemon and Liu (2010) apply RL to information presentation in a spoken dialogue system that gives restaurant recommendations to users. A particular focus is on
whether database hits should be summarised for the user, contrasted given the user’s preferences or whether a single recommendation should be given. An optimal action policy here depends both on the user’s preferences and the number of database hits. Similarly, Janarthanam and Lemon (2010) use RL to optimise NLG in troubleshooting dialogues where users are assisted in setting up a broadband connection. A special focus of this work is the fact that the user learns new jargon during the interaction with the system so that the learnt policy needs to be sensitive to a dynamic user model.

Reinforcement learning has also been applied to other natural language processing tasks (Branavan et al., 2009), which often use task completion as the primary component of their reward function and therefore require less or no simulation. In contrast, RL applications in dialogue or generation typically need to be trained in interaction with human users, which makes training more expensive. Even though simulated environments can be used, they often rely on linguistic or pragmatic features which may require annotation, depending on the domain. One possible solution to mitigate this problem has been to use Wizard-of-Oz data collections (Rieser and Lemon, 2008), which automatically log wizard actions and therefore can be used to bootstrap simulated environments from small data sets.

Research on reinforcement learning for NLG is in several ways related to planning. In particular, it is often seen as a possible solution to AI planning in which well-studied algorithms are used for finding action strategies for NLG tasks from a pre-defined set of knowledge and constraints. Please see (Koller and Petrick, 2011) for a recent survey of planning approaches to NLG. In contrast to other approaches, RL is particularly suited for tasks in which we are unsure of the best strategy to achieve a goal and wish the system to find an optimal policy automatically from interactions with the environment and user.

This article follows the general direction of the RL research discussed above by representing situated NLG as a sequential decision making problem that can be solved using trial and error search in an interactive context. In contrast to previous work, however, which has relied predominantly on flat RL, we formulate our NLG task as a hierarchical optimisation problem. This is often more scalable than flat RL settings and can be applied to larger search spaces, such as more complex generation scenarios than those that can be addressed using a flat RL setup.

### 2.2 Sharing knowledge across subtasks

A number of recent studies have presented evidence in favour of a joint treatment of subtasks by sharing knowledge among them. Angeli, Liang and Klein (2010) present a robust domain-independent NLG system that employs a joint treatment of content selection and surface realisation. In their approach, each generation decision is handled by a log-linear classifier that has access to all previous decisions and achieves better accuracy and human ratings than a system whose’s information is restricted to the local context. Lemon (2011) presents a joint optimisation approach to NLG and dialogue management in the area of information presentation. He shows that using reinforcement learning for the optimisation, a jointly optimised policy...
can learn when it is most advantageous to present information to the user or when to ask for more details to refine the query. In Cuayáhuítl and Dethlefs (2011b), we present a hierarchical reinforcement learning approach to spatially-aware dialogue management by optimising it jointly with route planning in a wayfinding domain. We show that the spatially-aware system—optimised jointly—generates the shortest possible route by adapting to individual users’ prior knowledge by guiding them past landmarks they are familiar with and avoiding junctions that cause confusion.

In addition to the studies discussed above, there have been suggestions for a joint treatment of syntax and semantics/discourse (Stone and Webber, 1998; Marciniak and Strube, 2004; Marciniak and Strube, 2005), of NLG and speech synthesis (Bulyko and Ostendorf, 2002; Nakatsu and White, 2006), of speech and gestures (Stone et al., 2004) and content planning and realisation (Bontcheva and Wilks, 2001). All of them have demonstrated that a joint treatment of interrelated tasks can significantly outperform its isolated counterpart. All of the joint architectures discussed above (Angeli, Liang, and Klein, 2010; Lemon, 2011; Cuayáhuítl and Dethlefs, 2011b) work essentially by making additional knowledge available to the components involved. Typically, this is knowledge that has traditionally been specific to one module of the system and is now shared between two or more modules in order to achieve a joint knowledge base on which to base decisions. These joint architectures have deliberately not attempted to share their full knowledge base, which would be computationally expensive. Instead, they have shared small parts of knowledge which were discovered from domain data or which the system designer expected to positively affect performance. In this way, they are computationally scalable and do not sacrifice the benefits of a modularised architecture.

A further approach to considering NLG decisions interdependently are systems like SPaRKy (Walker et al., 2007). Here, sentence generation takes an overgeneration and ranking approach. In a first step, a randomised set of alternative sentence plans is generated. In a second step, these are ranked according to a boosting score that predicts user ratings of the outputs. Joint decision making is possible in that an \(n\)-best list of alternatives is passed between modules, which can each be considered in the next module.

Here, we will follow the direction of sharing knowledge across generation subtasks so as to provide a richer context for decision making to our learning agent. In this way, the full utterance context can be considered rather than local context alone.

### 2.3 Situated Natural Language Generation

Related work on situated NLG has explored a range of different methods. Denis (2010) presents a rule-based approach to GIVE which works by systematically eliminating distractor buttons until a unique reference to a target object is possible. To achieve this, he makes use of the fact that referring expressions are not only determined by context but also modify it. Benotti and Denis (2011b) present an approach to GIVE based on corpus-based selection, which maps situations in the GIVE environment directly to human descriptions. This technique works with few or no annotations and therefore greatly reduces development costs. Also training
from unannotated data, Chen, Kim and Mooney (2010) present a system that learns to interpret and generate language based on pairs of action sequences and textual descriptions of RoboCup games. A particular challenge is that the action sequences are ambiguous in that not every action is described in the corresponding text. The authors’ best performing system in terms of surface realisation was optimised for precision by comparing generated system output against human-authored text. For content selection, the authors train their generator using a variant of the EM algorithm to estimate which events are worth including in a textual description and which are not.

Using supervised learning for situated generation, Stoia, Shockley, Byron and Fosler-Lussier (2006) use decision trees to learn content selection rules for noun phrases in a situated generation setting. Similarly, Dale and Viethen (2009) and Viethen, Dale and Guhe (2011) use decision trees to learn content selection rules for referring expressions in spatial settings. Garoufi and Koller (2011a) use a planning approach to make a first set of content selection decisions and then apply a maximum entropy model to resolve the remaining nondeterminacy with respect to surface realisation. All of these approaches have demonstrated that supervised learning is attractive for learning behaviour from a labelled corpus, discovering interdependencies between choices and performing decision making based on human behaviour. In contrast, based on the principle of assigning delayed rewards for a sequence of actions, reinforcement learning is typically well suited for optimising sequential decision making problems such as situated interaction. An example application is an NLG system that needs to generate an effective and coherent sequence of instructions. This principle is discussed in detail in Section 4.

3 Situated Natural Language Generation in the GIVE environment

Generation in situated settings typically requires the NLG system to adapt to changing circumstances in its physical environment, such as new objects and spatial configurations. In addition, we assume interaction with a constantly moving user, so that the system needs to monitor their progress and keep them on track.

3.1 Generating Instructions in Virtual Environments (GIVE)

The GIVE task (Generating Instructions in Virtual Environments) involves two participants, one instruction giver and one instruction follower, who engage in a ‘treasure hunt’ through a set of virtual worlds. The task can be won by finding and unlocking a safe and obtaining a trophy from it. It can be lost by stepping onto one of a number of red tiles and activating an alarm. To solve the task, the instruction giver has to guide the instruction follower in navigating through a world, and pressing a particular sequence of buttons. The sequence of buttons corresponds to a code that will, if pressed in the correct order, unlock the safe and release the trophy. There are also a number of distractor buttons present, though, which either have no effect or trigger an alarm. In the original GIVE task (Byron et al., 2009; Koller et al., 2010), the role of the instruction giver is taken by an NLG system of
the kind that we will develop in the remainder of this article. The NLG system’s action set includes navigation instructions, such as moving to the left/right, going straight, or leaving the room. The system also generates referring expressions, which need to be accurate in order to distinguish intended referents from their distractors. To do this, the virtual worlds also contain a set of landmarks, such as plants or furniture, which can be used as points of reference. The instruction follower, or user, is restricted to a number of non-verbal actions. They can either move to the front, left, right or back, or press a button. They can in addition ask for help by pressing a help button or cancel the game by pressing escape. Note that even though the user’s actions are confined to non-verbal behaviour, the task still resembles a dialogue setting in that the user is able to react to any instruction that the system produces. Figure 3 shows excerpts from three interactions between two humans during the GIVE task.

3.1.1 The GIVE-2 corpus

The GIVE-2 corpus (Gargett et al., 2010) is a collection of (63 English and 45 German) human-human dialogues on the GIVE task that was collected in a Wizard-of-Oz study to shed light on the strategies that human instruction givers employ when giving navigation instructions and referring expressions (REs) to their interlocutors. Participants in this scenario played three games in three different virtual worlds. After the first game, they switched roles for the last two games.

To facilitate the automatic analysis of the GIVE corpus dialogues and to provide our learning agent with information about the target domain, we annotated the English set of dialogues according to the annotation scheme shown with an example annotation in Figure 2. The annotations\(^1\) concern four areas: (1) the utterance itself and its type, (2) the semantic choices of a referring expression, where the set of spatial relations is taken from Bateman, Hois, Ross and Tenbrink (2010), (3) the spatial environment, i.e. the situational setting in which an instruction is produced, and (4) the user’s reaction to an instruction. The user reaction feature is key and will play an important role in training the learning agent in Section 5.2.

3.1.2 Instruction types in the human data

As an example of the task our NLG system faces, consider the instruction sequences of the GIVE corpus in Figure 3. All of these examples refer to the same situation, but instruction givers still employ a range of fundamentally different instruction giving strategies. Instructions differ in length, abstraction and semantic choices. We group them here into three types. Each type is characterised by a number of qualitative features discussed in the following.

The first instruction sequence guides the user by a high-level navigation strategy. It makes explicit reference to the dialogue history and to locations that the

Fig. 2. Sample annotation for a navigation instruction followed by a referring expression. Alternative annotation values are given in square brackets behind the actual values. This set of (possible) annotations defines our annotation scheme for the GIVE-2 corpus.

instruction follower has visited previously and is expected to remember (including how to get there). This strategy makes use of the structure of the environment by referring to doors, paths and rooms.

The second instruction sequence, in contrast, relies exclusively on guiding the instruction follower by low-level navigation. Every required action is explicitly verbalised and there is no reference to the environmental structure or dialogue history. High-level instructions represent contractions of low-level instructions.

The third instruction sequence, finally, lies in between the two extremes. While it takes advantage of the environmental structure and visual information, there are no references to the dialogue history. We call this mode of instruction giving mixed.

To design an NLG system that can solve the GIVE task, we will be concerned mainly with the generation of the following six instruction types.

- **Destination Instructions** aim to guide a user to their next subgoal in the virtual world, mainly by specifying the goal, rather than the way to the destination. An example is *Head back to the room with the plant.*

- **Direction Instructions** indicate changes of direction to the user, such as *Turn left at the door.*

- **Orientation Instructions** instruct the user to change their orientation. An example is *Turn 180 degrees left.*

- **Path Instructions** serve to guide the user along a certain path, as in *Follow the corridor until you reach a door.*

- **Instructions to go straight** aim to guide the user to go straight. An example is *Keep going straight.*
Referring Expressions are instructions to press a particular button, for example, *Push the red button to the left of the yellow.*

The hierarchy of learning agents will make decisions at different level of granularity to contribute to the generation of these six instruction types. While the agent’s knowledge is partially informed by the annotations of the GIVE corpus, it is also informed by linguistic knowledge that was obtained through manual analysis of the domain. Note however that the route plan is provided by the GIVE client,\(^2\) which informs the NLG system about the next (sub-)goal and about how to get there. It also provides information about the user’s location, spatial objects and visibility.

In the end, though, the learning agent has to decide how much detail to provide to the user and whether to realise route plans step-by-step or all at once.

4 A hierarchical optimisation approach for language generation

A central characteristic of RL-based approaches is that they typically specify abstract system goals, such as help the user set up the broadband connection without using words they do not understand and without unnecessary descriptions (Jahnarthanam and Lemon, 2010), or help the user find a restaurant they like without presenting every possible option to them, but still give them a good overview of the choices (Rieser, Lemon, and Liu, 2010). The system is always just told what to achieve, but not how to achieve it. It is then the learning agent’s objective to try different strategies and discover the best. For our situated NLG task, we could say that we wish the agent to guide the user to the nearest navigation (sub-)goal, e.g. the next button to press, so that they get there as quickly as possible and obtain the trophy with as few problems and confusions as possible.

4.1 Reinforcement learning

The goal of an RL agent is to map situations to actions in a goal-directed manner so as to maximise a long-term, numeric reward signal. The computational model underlying RL agents is the Markov Decision Process, or MDP (Sutton and Barto, 1998). A standard MDP can be defined formally as a four-tuple \( <S, A, T, R> \).

- \( S = \{s_0, s_1, s_2, \ldots, s_N\} \) is a set of states that summarise all information, present and past, that the agent needs in order to behave in its world of situations. It includes, for example, the status of the environment, such as present objects and buttons, the user’s state of confusion or the next navigation action to execute. States must allow the agent to monitor its progress in the learning task at any time and observe the effects that its actions have. Thus whenever the agent takes an action \( a \) in state \( s \) at time step \( t \), the updated state \( s_{t+1} = s' \) (at time step \( t + 1 \)) should represent the action’s effect on the environment. In this way, the agent is able to learn from its experience.

- \( A = \{a_0, a_1, a_2, \ldots, a_M\} \) is the set of actions available to the agent. It defines the agent’s behavioural potential and forms the basis for decision making and the principle of learning from trial and error. Example actions include generating instructions such as turn left, mentioning the colour of a referent or telling the user to stop.

- \( T \) is a probabilistic state transition function indicating the next state \( s' \) from the current state \( s \) and the action \( a \). It represents the way in which an action changes the current state of the world. \( T \) is represented by a conditional probability distribution \( P(s'|s, a) \) satisfying \( \sum_{s' \in S} P(s'|s, a) = 1, \forall (s, a) \). For example, if the user has to press a particular button, this will be represented with probability \( p \) for the state transition to the state with the right button pressed, and probability \( 1 - p \) for transitioning to a different state due to a wrong action (such as a wrong button pressed).
Hierarchical Reinforcement Learning for Situated NLG

- $R$ is a reward function $R(s'|s, a)$ specifying a numeric reward that an agent receives for taking action $a$ in state $s$. Rewards allow the agent to evaluate its decision making process. The reward at time $t+1$ is also denoted by $r'$. Rewards provide the primary feedback mechanism for the agent.

The dynamics of an MDP can be described as follows. At the beginning of an interaction between the agent and the environment, when the time step $t = 0$, the agent receives a representation of the current situation, called the state $s_t \in S$. It needs to perform an action $a_t \in A$. As a result, the agent will receive a reward $r_{t+1} \in R$ and observe the next state $s_{t+1} \in S$, which is the updated environment state. This process can be seen as a finite sequence of states, actions and rewards \{s_0, a_0, r_1, s_1, a_1, ..., r_{t-1}, s_t\}. Any mapping from states to actions is called a policy. Ultimately, the agent’s goal is to learn an optimal policy denoted by $\pi^*$, a mapping from every state $s$ to an action $a$ that will yield the highest expected return. An optimal policy can be found according to

$$\pi^*(s) = \arg\max_{a \in A} Q^*(s, a),$$  

where $Q^*$ is the function of expected rewards for executing action $a$ in state $s$ and then following $\pi^*$. For learning single-task NLG policies using flat RL, such a function can be found using algorithms such as SARSA (Sutton, 1996) or Q-Learning (Watkins, 1989), among others. See Sutton and Barto (1998) or Szepesvari (2010) for a detailed account of the RL paradigm.

### 4.2 Hierarchical reinforcement learning

RL systems with large state spaces are affected by a problem referred to as the curse of dimensionality, the fact that state spaces grow exponentially with the number of state variables they take into account. When the state space grows too large, the agent will not be able to find an optimal policy for a task, which affects its practical application in large systems (such as many real-world systems or the one we are designing for GIVE). The best one can do in such situations is to provide an approximate solution, such as a divide-and-conquer approach to optimisation. For this, we divide the generation task into several subtasks, which have smaller state spaces and can therefore find a solution more easily. In other words, we learn a hierarchy of policies for generation subtasks, rather than learning one single policy for the whole task. An alternative way of dealing with the curse of dimensionality is to use function approximation techniques (Henderson, Lemon, and Georgila, 2005; Pietquin et al., 2011; Jurcícek, Thompson, and Young, 2011), which are not guaranteed to converge to optimal policies, though.

Any flat learning agent that is characterised by a single MDP can be decomposed into a set of subtasks $M_i^j$, where $i$ and $j$ are indexes that uniquely identify each subtask in a hierarchy of subtasks such that $\mathcal{M} = \{M_0^0, M_1^0, M_1^1, M_2^1, ..., M_X^Y\}$. These indexes do not specify the order of execution of subtasks, because the order of execution is subject to learning. Each subtask, or agent in the hierarchy, is defined as a Semi-Markov Decision Process (or SMDP) $M_j = (S_j^i, A^i, T_j^i, R_j^i)$, in which $S_j^i$
\( = \{s_0, s_1, s_2, \ldots, s_N\} \) is a set of states of subtask \( M_j \). \( A^i_j = \{a_0, a_1, a_2, \ldots, a_M\} \) is a set of actions of subtask \( M_j \) that can be either primitive or composite. Primitive actions are single-step actions as in an MDP and receive single rewards. Composite actions are temporally-extended actions that correspond to other subtasks in the hierarchy and are children of the current, their parent, subtask, such as referring expression generation. Composite actions receive cumulative rewards.

The execution of a composite action, or subtask, takes a variable number of time steps \( \tau \) to complete which is characteristic of the SMDP model (and distinguishes it from an MDP). The parent SMDP of a subtask passes control down to its child subtask and then remains in its current state \( s_t \) until control is transferred back to it, i.e. until its child subtask has terminated execution. It then makes a transition to the next state \( s' \). \( T^i_j \) is a probabilistic state transition function of subtask \( M^i_j \), and \( R^i_j \) is a reward function \( R^i_j(s', \tau | s, a) \) for subtask \( M^i_j \) that specifies the reward that the agent receives for taking action \( a \in A^i_j \) (lasting \( \tau \) time steps) and making a transition from state \( s_t \) to state \( s_{t+\tau} \in S^i_j \). Discounted cumulative rewards of composite actions are computed according to \( r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots + \gamma^{\tau-1} r_{t+\tau} \), where \( \gamma \) is called the discount rate, a parameter which is \( 0 \leq \gamma \leq 1 \) and indicates the relevance of future rewards in relation to immediate rewards. As \( \gamma \) approaches 1, both immediate and future rewards will be increasingly equally valuable. The equation for optimal hierarchical action selection is

\[
\pi^*_{j}(s) = \arg \max_{a \in A^i_j} Q^*_{i j}(s,a),
\]

where \( Q^*_{ij}(s,a) \) specifies the expected cumulative reward for executing action \( a \) in state \( s \) and then following \( \pi^*_{i j} \). For learning hierarchical NLG policies, we use the HSMQ-Learning algorithm (Dietterich, 2000), a hierarchical version of Q-Learning. During policy learning, \( Q \)-values are updated according to the following update rule (Sutton and Barto, 1998, p. 37)

\[
\text{NewEstimate} \leftarrow \text{OldEstimate} + \text{StepSize} \left[ \text{Target} - \text{OldEstimate} \right].
\]

Using the notation above, this corresponds to

\[
Q^i_j(s,a) \leftarrow Q^i_j(s,a) + \alpha \left[ r + \gamma^{\tau} \max_{a'} Q^i_j(s',a') - Q^i_j(s,a) \right],
\]

where \( \alpha \) is a step-size parameter. It indicates the learning rate which decays from 1 to 0, for example as in \( \alpha = 1/(1 + \text{visits}(s,a)) \), where \( \text{visits}(s,a) \) corresponds to the number of times that the state-action pair \( (s,a) \) has been visited previous to time step \( t \). Please see Cuayáhuítl (2009), p. 92, for its application to spoken dialogue management, and Dethlefs and Cuayáhuítl (2010) for an application to NLG besides this journal.

### 5 Training and learning setting

Section 4 has provided an abstract description of hierarchical RL, which we will now apply to situated NLG. We will first design the state and action space for our hierarchical reinforcement learner for the GIVE task. This will be a linguistically-informed knowledge engineering task. We will then define a simulated environment
and reward function and train the hierarchical learner in a set of training navigation worlds.

5.1 The hierarchy of learning agents interacting with the environment

This section will provide details of the knowledge engineering involved in applying hierarchical RL to GIVE. We first explain how the learning agent interacts with its environment during training (and execution) and then define a hierarchy of learning agents specifically for GIVE.

5.1.1 Interaction with the environment

An illustration of the agent-environment interaction, as required during learning or execution of the learning agent, is shown in Figure 4. The agent’s behaviour, represented by the upper box, is following a policy $\pi^*$, which indicates the best action for a given state at time $t$, $a_t^m = \pi^*(s_t^m)$. Here, $m$ stands for machine. This action is passed to the generation environment, where its effects on the user and the virtual world are observed and represented in the updated state $s_{t+1}^m$. Interaction with the generation environment is the main contributor to the agent’s learning process. It contains three types of information: information concerning the knowledge base, the virtual world and the user. The agent’s knowledge base contains all knowledge held by the agent about the virtual world, the user and the current generation state and history. From here, knowledge is also distributed to different learning agents and enters their state representation. The virtual world contains objects of the world, such as buttons and objects as well as the user’s concrete
position and angle in the world. During training, this knowledge is estimated from the simulated environment (see Section 5.2), during execution it is taken directly from the GIVE environment and planner. Knowledge of the virtual world is passed to the agent’s knowledge base as the world state $w_t$ so that it can be taken into account for action selection. In return, the current agent state $s_t^m$ is passed back to the virtual world so that it can be taken into account for updates to the world. The user’s knowledge base contains all knowledge about the virtual world that the user has gained. For example, if the user has pressed a certain button or visited a particular room previously, we assume that the user is now familiar with these objects. Such user knowledge can only be estimated since we can never be certain about the user’s knowledge. The simulated user behaviour is the agent’s main way of learning about the user’s current state, such as whether the user is confused or not, and to evaluate its own action policies. User behaviour is classified into four actions: perform desired action, perform undesired action, wait and request help. Since the user cannot communicate verbally in GIVE, this limited action repository provides a sufficient notion of the user’s state. The user state $s_t^u$ is passed to the action simulator from the user’s knowledge base so that actions can be estimated based on the user’s knowledge. User actions $a_t^u$ produced by the simulator (or the actual user during a game) are communicated back to the knowledge base as updates.
Fig. 6. This hierarchy of state-action sequences shows the high-level dynamics of states, actions and transitions for an example generation episode for a high-level navigation instruction. The empty circles represent generation states (s0=initial state, s13=final goal state), the dark (blue) circles represent composite actions, the light (green) circles represent primitive actions, and the dotted arrows represent state transitions across learning agents. The low-level details of this example are given in Appendix B.

5.1.2 The hierarchy of learning agents

As a more concrete description of how knowledge and actions are passed between agents, Figure 5 shows the hierarchy of learning agents we designed for the GIVE task. It comprises 14 different agents whose policies can be roughly categorised as tasks of content selection (\(\pi_0^0, \pi_1^0, \pi_2^0, \pi_1^1, \pi_2^1, \pi_3^1\) and \(\pi_4^1\)), utterance planning (\(\pi_2^2\)) and surface realisation (\(\pi_3^2, ..., 5\)). Note that information is always passed between learning agents in the form of state updates that follow user or system actions.

**Content selection** is responsible for all semantic decisions made by the learning agent, such as whether to choose a high- or low-level navigation strategy, whether to mention a referent’s colour or not, etc. **Utterance planning** focuses on how to organise semantic content into a distinct set of messages. For example, should a set of instructions be aggregated or presented separately, what thematic structure should be used, etc. **Surface realisation** finally chooses a realisation for the utterance from a set of candidates (Section 5.3) for our six instruction types. For a joint optimisation, these 14 agents would share certain knowledge variables among them. This shared knowledge is pre-defined by the system designer and gives us the opportunity to optimise subtasks jointly rather than in isolation. It allows the learning agents to consider different types of decisions interdependently that affect the trade-off between detail and efficiency in situated interaction. At the same time, it preserves the benefits of a modular architecture.

Generation always begins with the root agent $M_0^0$ (indexed by its policy $\pi_0^0$) which has the option of taking primitive actions or invoke composite actions of reference or navigation. In the latter case control is passed to a child subtask, agent $M_0^1$ for reference or agent $M_1^1$ for navigation, respectively. The flow of control is indicated by the arrows in Figure 5. During the process of generating an utterance, control is passed between agents, such as from parent to child when a subtask is called, and from child back to parent once a subtask has terminated. Whenever control is transferred back to the root agent, an episode has been completed and execution terminates. One episode (from state $s_0$ to state $s_T$) corresponds to one utterance. Figure 6 illustrates the passing of control between agents during a generation episode. In this case, a destination instruction is generated, which uses a high-level navigation strategy. In addition, an utterance plan is needed which specifies how the instruction fits in with other instructions.

While this figure only provides a high-level example, please see Appendix B for all details and individual actions and state transitions. The complete state-action space of the hierarchical learning agent has a size of $\sum_{i,j} |S_i^j \times A_j^i| = \sum_{i,j} \left[ \prod_{k(i,j)} |f_{k(i,j)}| \times |A_j^i| \right] = 1,480,869.4$ Here, $(i,j)$ represents an agent in the hierarchy, $f_{k(i,j)}$ represents the feature set of agent $(i,j)$ and $k$ refers to features $k$ in agent $(i,j)$. In contrast, a flat agent using the same states and actions would have the (very large) state-action space of $|S \times A| = \left( \prod_k |f_k| \right) \times |A| = 3 \times 10^{57}$, indicating the advantage of using a hierarchical decomposition for more scalable decision making. The complete state-action space of the hierarchical agent (and the pre-specified shared knowledge variables for a joint optimisation) are given in Appendix A.

5.2 The simulated environment

Typically, a reinforcement learning agent needs to be exposed to a large number of interactions during training to learn an optimal policy. Since it is impractical to use real users for these interactions, we use a simulated environment instead and estimate it from our annotations of the GIVE corpus. Our goal is to simulate different spatial surroundings in which the agent can try a multitude of action strategies in order to learn an optimal one by trial and error. The effect of each action will be simulated in the form of a user reaction from among $Y = \{\text{perform desired action}, \text{perform undesired action}, \text{wait}, \text{request help}\}$. Users in our training data were generally cooperative so that a good system action strategy always results

---

4 The detailed calculation involves computing the sum over all possible state-action pairs per agent. For the agents specified in Appendix A, this is $(2 \times 2 \times 5 \times 5 \times 3 \times 5) + (2 \times 2 \times 3 \times 5 \times 2 \times 2 \times 2 \times 2 \times 5) + (2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 3 \times 5 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 14) + (2 \times 2 \times 2 \times 2 \times 4 \times 4 \times 5 \times 9 \times 3 \times 18) + (3 \times 2 \times 2 \times 2 \times 3 \times 2 \times 5) + (2 \times 3 \times 2 \times 2 \times 2 \times 3 \times 2 \times 3 \times 2 \times 6) + (3 \times 2 \times 2 \times 2 \times 4 \times 2 \times 2 \times 2 \times 4) + (4 \times 7 \times 4 \times 8 \times 3 \times 19) + (3 \times 5 \times 4 \times 7 \times 3 \times 15) + (5 \times 4 \times 3 \times 5 \times 9) + (5 \times 7 \times 7 \times 7 \times 3 \times 23) + (6 \times 5 \times 5 \times 3 \times 13) + (2 \times 2 \times 3 \times 4 \times 2 \times 5 \times 2 \times 2 \times 3 \times 10) + (2 \times 2 \times 2 \times 3 \times 3)$, in the order in which the agents appear in Appendix A. In the non-hierarchical RL case, numbers have to be multiplied instead of summed, because no hierarchical decomposition applies.
in the user performing the desired action. All other user reactions indicate a non-optimal system action.

Our simulated environment is based on two Naive Bayes classifiers, one for simulating user reactions $Y$ (the classes) to referring expressions and one for simulating user reactions to navigation instructions. We use two separate classifiers rather than one because different feature sets are relevant for each system action type. For simulating user reactions to referring expressions, we use the following features $X$:

- **discriminating colour referent** $x_0 = \{\text{true, false}\}$, indicates whether the referent’s colour is uniquely identifying or not,
- **discriminating colour distractor** $x_1 = \{\text{true, false}\}$, indicates whether any of the distractor’s colours are uniquely identifying or not,
- **number of distractors** $x_2 = \{0, 1, 2, 3, \text{more}\}$, indicates the number of distractors present, if any,
- **number of landmarks** $x_3 = \{0, 1, 2, 3, \text{more}\}$, indicates the number of landmarks present, if any,
- **is visible and near** $x_4 = \{\text{true, false}\}$, indicates whether the referent button is near and visible to the user (the conditions to press a button),
- **referent colour mentioned** $x_5 = \{\text{true, false}\}$, indicates whether the system’s instruction included the colour of the referent, and
- **within dialogue history** $x_6 = \{\text{true, false}\}$, indicates whether the button is already in the dialogue history, e.g. because it has been pressed before.

For simulating user reactions to navigation instructions, we use the following features $Z$:

- **number of landmarks** $z_0 = \{0, 1, 2, 3, \text{more}\}$, indicates the number of landmarks present, if any,
- **is visible and near** $z_1 = \{\text{true, false}\}$, indicates whether the button is visible and near (or whether we need to navigate further towards it),
- **navigation level** $z_2 = \{\text{high-level, low-level}\}$, indicates whether the system’s instruction was a high- or low-level type instruction,
- **navigation content** $z_3 = \{\text{destination, direction, orientation, path, straight}\}$, indicates the type of navigation instruction generated, and
- **within dialogue history** $z_4 = \{\text{true, false}\}$, indicates whether the next target (a button, room or other object) is already in the dialogue history.

Using these feature sets, we predict user reactions from our annotations of the GIVE corpus by sampling from the distribution $P(Y|X)$ for referring expressions, and by sampling from $P(Y|Z)$ for navigation instructions. All features describing the environment, such as the number of buttons or landmarks present, were simulated from unigram language models estimated from the GIVE corpus. These features were simulated with the same distribution as they occur in the GIVE corpus, but deliberately so that the agent would encounter as many different settings as possible and not be restricted to the GIVE worlds$^5$ shown in Figure 7.

---

$^5$ The worlds of the GIVE corpus, used for training, can be downloaded from [http://www.give-challenge.org/research/page.php?id=software](http://www.give-challenge.org/research/page.php?id=software) [accessed April 22, 2013].
Fig. 7. Illustration of the training worlds that are the basis of the simulated environment. All worlds require skills for navigation and disambiguation at a medium level of difficulty.

To train our classifiers, we used the Weka toolkit (Witten and Frank, 2005), and evaluated our classifiers in a ten-fold cross validation. For referring expressions, our classifier achieved an accuracy of 78% and for navigation instructions an accuracy of 86%, yielding an average of 82%. As a baseline, a ZeroR (majority class) classifier yields an average accuracy of 69% by always voting for the most likely option.

5.3 A three-dimensional reward function

We use a reward function with three dimensions for optimisation: (1) one for achieving maximal user satisfaction, (2) one for rewarding human-like surface realisation decisions and (3) one for optimising the proportion of alignment and variation in system utterances. Each of them will be discussed in turn.

5.3.1 Dimension 1: User satisfaction

The first dimension aims to maximise user satisfaction. According to the PARADISE framework (Walker et al., 1997; Walker, Kamm, and Litman, 2000), the performance of a (spoken) dialogue system can be modelled as a weighted function of task success and dialogue cost measures (e.g., number of turns, interaction time, etc.). We argue that PARADISE is also useful to assess the performance of an interactive NLG system, since both objective measures (e.g., task success) and subjective measures (e.g., ease of understanding) seem equally relevant for NLG systems in situated contexts. To identify the strongest predictors of user satisfaction (US) in situated dialogue and NLG systems, we performed an analysis of subjective and objective dialogue metrics collected with an indoor wayfinding system, based on PARADISE (Dethlefs et al., 2010). We used a graded task success (GTS) metric (Tullis and Albert, 2008), rather than a binary (success=1 / failure=0) metric, so as to be more sensitive to problems that users experienced during navigation. This metric assigns different numerical values depending on the problems that users en-
Hierarchical Reinforcement Learning for Situated NLG

It is defined as follows, where FTL means “finding the target location”.

\[
GTS = \begin{cases} 
1 & \text{for FTL without problems} \\
2/3 & \text{for FTL with small problems} \\
1/3 & \text{for FTL with severe problems} \\
0 & \text{other.} 
\end{cases}
\]

In order to identify the relative contribution that different factors have on the variance found in user satisfaction scores, we performed a standard multiple linear regression analysis on our data. Results revealed that the metrics ‘user turns’ and ‘graded task success’ were the only predictors of user satisfaction at \( p < 0.05 \). The binary task success metric was not significant \( (p < 0.39) \). Based on this, we ran a second analysis using only those variables that were significant predictors in the first regression analysis, i.e. graded task success and the number of user turns (which are negatively correlated). We obtained the following performance function:

\[
\text{Performance} = 0.38 \bar{N}(GTS) - 0.87 \bar{N}(UT),
\]

where 0.38 is a weight on the normalised value of \( GTS \) and 0.87 is a weight on the normalised value of \( UT \) (the number of user turns).\(^7\) Using this reward function, our learning agent is rewarded for short interactions (as few user turns as possible) at maximal (graded) task success. User turns correspond to user reactions following system instructions. If the user reacts positively (carries out the instructions), task success is rated with 1; if they hesitate once, it is 2/3, if they hesitate more than once it is 1/3 and if they get lost (carry out a wrong action), it is 0. In this way, the agent receives the highest rewards for the most efficient utterance followed by a positive user reaction. This reward function is used by all agents \( M_0 \ldots M_4 \) dealing with content selection and utterance planning. Rewards are assigned after each system instruction and the user’s reaction (i.e. whenever an agent of \( M_0 \ldots M_4 \) has reached its goal state). The learning algorithm propagates this reward back to all agents that contributed to the decisions that led to the generated instruction.

5.3.2 Dimension 2: Naturalness

The second dimension focuses on surface realisation performed by agents \( M_3 \ldots 5 \). We have decided to base surface realisation decisions based on probabilities of surface forms as they occur in the GIVE corpus and use these probabilities as rewards to inform the agent’s learning process. While in this particular case we use Bayesian Networks to represent probabilistic generation spaces per instruction type (for destination, direction, orientation, path, ‘straight’ and referring expression), nothing depends on the model chosen. Any surface realiser that is able to return a probability given a surface form would be suitable, including \( n \)-gram language models.

\(^7\) We normalised all values to account for the fact that they can be measured on different scales according to \( \bar{N}(x) = \frac{x - \bar{x}}{\sigma_x} \), where \( \sigma_x \) corresponds to the standard deviation of \( x \).
Please see Dethlefs and Cuayáhuítl (2011) for the details of how our Bayesian Networks were trained and Dethlefs and Cuayáhuítl (2012) for a comparison with other graphical models.

For generating natural surface forms, the agent’s rewards will be based on the probability of the word sequence it has generated. This means that having generated word sequence \( w_0 \ldots w_n \), it will receive the probabilistic reward \( Pr(w_0 \ldots w_n) \). In Bayesian Networks, this reward can be obtained through probabilistic inference, according to

\[
\text{Surface String Probability} = Pr(w_1 \ldots w_n|e),
\]

where \( w_1 \ldots w_n \) refer to individual words, and \( e \) can correspond to non-linguistic context derived from the interaction history. For example, if we wanted to compute the probability of the sentence “go to the sofa”, this can be expressed as \( Pr(\text{verb}=\text{go}, \text{prep}=\text{to}, \text{relatum}=\text{the sofa}|e) \).

5.3.3 Dimension 3: Balancing alignment and variation

The third dimension of the reward function aims to balance the proportion of alignment and variation in a natural and human-like fashion. It is used by the surface realisation agents \( M^3_0 \ldots M^3_5 \). From the human GIVE data, it was observed that instruction givers tend to self-align with their own utterances and vary them in an about equal fashion. An example of this is provided in Table 1. The aligned phrases here are shown in bold-face and the number of instructions intervening between aligned instructions are given in parentheses. In the first example, the instruction giver uses the phrase “you want” with high frequency and across instruction types. The phrase per se has a rather low frequency in the corpus on the whole (1.8% of all verbs). In the second example, the instruction giver produces referring expressions almost exclusively using the verbs “click” (33.3% in this dialogue and 33% in the entire corpus) and “hit” (66.6% in this dialogue, 6.6% in the corpus). We can see that human instruction givers do not only self-align with their own utterances, but they also introduce a significant amount of variation, possibly to reduce the repetitiveness of their utterances.

We will not investigate the question here of why variation (or alignment) occurs in human discourse, but see Levelt (1989), Belz and Reiter (2006) and Foster and Oberlander (2006) for some hypotheses. Rather we will take the stance that if it occurs as ubiquitously as we have observed in our human data, then it should be part of the agent’s learning objectives. Therefore, we define a constituent alignment score (CAS) which indicates the proportion between alignment and variation for each constituent in the discourse. It is computed as \( \text{CAS} = \frac{\text{Lexical Tokens in Discourse}}{\text{Total Number of Tokens}}, \) which yields a number in the range of \([0 \ldots 1]\). Please see Dethlefs and Cuayáhuítl (2010) for details of this computation and its background.

We would like our agent to generate utterances so that the CAS for each utterance is as close to 0.5 as possible. To achieve this, we assign each generated utterance a probabilistic reward sampled from a Gaussian distribution. In probability theory this has a probability density function defined as

\[
f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}},
\]

where \( \mu \)
Hierarchical Reinforcement Learning for Situated NLG

Examples of Alignment in the GIVE Corpus

(1) Lexical Alignment Across Instruction Types

... (15 instructions) ...
great, **you want** to press that green button ... (1 instruction) ...
you **want** to press the yellow on the wall to the left first ... (3 instructions) ...
you **wanna** get that red button ... (1 instruction) ...
now you **want** to get the blue button ... (9 instructions) ...
you **want** to exit the room you are in ... (17 instructions) ...
you **want** to keep going straight but to the left ... (25 instructions) ...
okay you **want** to take a left ... (13 instructions) ...

(2) Lexical Alignment in Referring Expressions

ok, **hit** the blue button on the wall behind you ... (11 instructions) ...
**click** the green button ... (1 instruction) ...
**click** the yellow button on the wall
now **click** the red button directly behind you ... (1 instruction) ...
now **hit** the blue one directly next to the yellow one
ok, **hit** the other red button, closer to the opening ... (4 instructions) ...
**hit** the green button near the couch ... (1 instruction) ...
and **hit** the red button ... (4 instructions) ...
**hit** the yellow button ... (7 instructions) ...

Table 1. Examples of (self-)alignment in the GIVE corpus. In the first example, the instruction giver uses the phrase you want with high frequency and across instructions. In the second example, the instruction giver uses exclusively the verbs click and hit in their referring expressions. The number of intervening instructions are shown in parentheses behind each instruction.

refers to the mean and $\sigma^2$ refers to the variance. The right-hand side of this equation is also commonly denoted as $\mathcal{N}(x|\mu, \sigma^2)$ so that the probability density function that we use for the sampling of rewards can be defined as:

$$P(CAS) \approx \mathcal{N}(CAS|\mu, \sigma^2),$$

where in our case we used a mean of $\mu = 0.5$, a variance of $\sigma = 0.2$. A CAS score in the range $[0 \ldots 1]$ indicates the proportion of alignment and variation.
5.3.4 Bringing all dimensions together

For the final experiments, we can bring all dimensions of the reward function together by summing rewards whenever more than one applies. For example, at the end of an utterance (upon reaching the goal state), usually the reward for the Performance of the utterance will apply, the reward for the Surface_String_Probability and the reward for Alignment_Variation. Accordingly, the reward for the utterance can be computed as

\[
\text{Reward} = \text{Performance} + \text{Surface_String_Probability} + P(\text{CAS}). \tag{8}
\]

For all dimensions and agents, a reward of -1 is assigned for every action in the hierarchy so as to prevent the agent from choosing actions multiple times and entering into loops. For example, it could happen that an agent chooses an action repeatedly that has yielded a positive reward in the past (such as choosing a surface realisation for the verb), even though it does not change the state of the environment anymore and instead fails to take other relevant actions (such as choosing a surface realisation for the direction). A small negative reward for repeated actions that do not change the state of the environment can therefore prevent such loops.

6 Evaluation

In this section, we will evaluate our hierarchical learning framework both in simulation and in a human evaluation study. We will focus particularly on a comparison of a joint generation policy, with shared knowledge, and an isolated generation policy. A brief comparison with state-of-the-art approaches for GIVE is also provided.

6.1 Simulation-based evaluation

Using simulation, we have trained two policies, a joint policy and an isolated policy. A qualitative analysis after 150 thousand training episodes reveals the following learnt behaviour. Figure 8 compares the average rewards (averaged over ten runs) of (a) a jointly optimised policy, i.e. using shared knowledge, and (b) an isolated policy, using no shared knowledge. We can see that a joint optimisation achieves higher overall rewards over time. An absolute comparison of the average rewards (rescaled from 0 to 1) of the last 1000 training episodes of each policy shows that the joint behaviour improves the isolated behaviour by 34% ($p < 0.0001$).

The joint policy has learnt to prefer high-level navigation over low-level navigation, but switch the navigation strategy when the user gets confused. It uniquely identifies a referent button by preferring the use of a discriminating colour, and otherwise (if neither the referent nor a distractor has a discriminating colour) use either a spatial relation, a distractor or a landmark (in this order of preference). If a distractor is used, the referent is located in relation to it, such as Press the yellow

---

This relies on the assumption that all rewards have equal weight with respect to the overall performance. Experimentation with different weights are left for future work.
Hierarchical Reinforcement Learning for Situated NLG

Fig. 8. A comparison of average rewards attained by the joint policy, using shared knowledge, and the isolated policy, using no shared knowledge, over time.

Table 2. Sample dialogue using the jointly optimised policy. The system starts using a high-level navigation strategy, and then switches to low-level as the user gets confused. Eventually it switches back to high-level.

<table>
<thead>
<tr>
<th>No.</th>
<th>Conversant</th>
<th>Utterance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>USR</td>
<td>starts game: ‘How do I get to the trophy?’</td>
</tr>
<tr>
<td>2</td>
<td>SYS</td>
<td>Go until the end of the hall and turn to the right.</td>
</tr>
<tr>
<td>3</td>
<td>USR</td>
<td>hesitates (i.e., waits for more than four seconds)</td>
</tr>
<tr>
<td>4</td>
<td>SYS</td>
<td>Turn 90 degrees right, then go straight.</td>
</tr>
<tr>
<td>5</td>
<td>USR</td>
<td>executes instructions</td>
</tr>
<tr>
<td>6</td>
<td>SYS</td>
<td>Now go until the end of the corridor and turn left.</td>
</tr>
<tr>
<td>7</td>
<td>USR</td>
<td>executes instructions</td>
</tr>
<tr>
<td>8</td>
<td>SYS</td>
<td>Good! Now walk to the red button left of the blue.</td>
</tr>
<tr>
<td>9</td>
<td>USR</td>
<td>executes instructions</td>
</tr>
<tr>
<td>10</td>
<td>SYS</td>
<td>Click the red button.</td>
</tr>
<tr>
<td>11</td>
<td>USR</td>
<td>performs desired action: presses red button</td>
</tr>
<tr>
<td>12</td>
<td>SYS</td>
<td>Very good.</td>
</tr>
</tbody>
</table>

button beside the blue. In addition, it will use composite presentations for at most two instructions (and aggregate them) and incremental displays otherwise. It has learnt to use temporal markers for more than three instructions. Finally, the agent has learnt to balance the trade-offs of variation and alignment while still acting in accordance with the language model.

Tables 2 and 3 show example interactions (from simulation) with the joint and isolated policy, respectively. These dialogues illustrate the importance of graded task success: while both users are successful in the end, the user of the jointly
Table 3. Sample dialogue using the isolated policy. The agent prefers efficient instruction giving. It uses high-level navigation and aggregates utterances where possible. In terms of repair, only the immediately preceding utterance is repaired.

<table>
<thead>
<tr>
<th>No.</th>
<th>Conversant</th>
<th>Utterance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>USR</td>
<td>starts game: ‘How do I get to the trophy?’</td>
</tr>
<tr>
<td>2</td>
<td>SYS</td>
<td>Go to the end of the hall and then turn left.</td>
</tr>
<tr>
<td>3</td>
<td>USR</td>
<td>hesitates (i.e., waits for more than four seconds)</td>
</tr>
<tr>
<td>4</td>
<td>SYS</td>
<td>Turn 90 degrees right, go straight and go down the corridor. Then turn left.</td>
</tr>
<tr>
<td>5</td>
<td>USR</td>
<td>turns right, requests help</td>
</tr>
<tr>
<td>6</td>
<td>SYS</td>
<td>Go to the end of the hall and then turn left.</td>
</tr>
<tr>
<td>7</td>
<td>USR</td>
<td>undesired action: turns left</td>
</tr>
<tr>
<td>8</td>
<td>SYS</td>
<td>Wait! Go back, then straight and then follow the corridor.</td>
</tr>
<tr>
<td>9</td>
<td>USR</td>
<td>executes instructions</td>
</tr>
<tr>
<td>10</td>
<td>SYS</td>
<td>Very good. Now go to the red one left of the blue and click the button.</td>
</tr>
<tr>
<td>11</td>
<td>USR</td>
<td>undesired action: presses blue button</td>
</tr>
<tr>
<td>12</td>
<td>SYS</td>
<td>No, not this one! I meant the other button. Click it.</td>
</tr>
<tr>
<td>13</td>
<td>USR</td>
<td>hesitates (i.e., waits for more than four seconds)</td>
</tr>
<tr>
<td>14</td>
<td>SYS</td>
<td>Click the red.</td>
</tr>
<tr>
<td>15</td>
<td>USR</td>
<td>performs desired action</td>
</tr>
<tr>
<td>16</td>
<td>SYS</td>
<td>Very good.</td>
</tr>
</tbody>
</table>

optimised dialogue is likely to have a substantially higher user satisfaction than the user interacting with the isolated system. We can also see that utterances in the isolated case are on average longer and seem to balance efficient instruction giving and the user’s cognitive load less optimally than the joint policy.

6.2 Task-based evaluation

In this section, we compare our jointly optimised policy with a policy optimised in isolation in a human evaluation study. We formulate the hypothesis that the sharing of knowledge across generation subtasks can lead to more successful interactions with fewer problems that are more positively perceived by human users.

6.2.1 Experimental methodology

We use objective and subjective metrics based on the PARADISE framework (Walker et al., 1997) for evaluating dialogue systems in order to evaluate our systems for the GIVE task. Table 4 gives an overview of the objective metrics that we use to evaluate the two system versions, jointly optimised and optimised in isolation. Under the category interaction efficiency, we find metrics such as the time that an interaction took, the number of system turns and system words, and the number of user turns (we count as user turns help requests or hesitations that last
Table 4. Objective metrics that were logged during interactions.

<table>
<thead>
<tr>
<th>Objective Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Interaction Efficiency</strong></td>
<td></td>
</tr>
<tr>
<td>(O1) Elapsed time (mm:ss)</td>
<td>How long was the interaction?</td>
</tr>
<tr>
<td>(O2) System turns</td>
<td>How many system turns?</td>
</tr>
<tr>
<td>(O3) System words</td>
<td>How many system words overall?</td>
</tr>
<tr>
<td>(O4) System words per turn</td>
<td>How many system words per turn?</td>
</tr>
<tr>
<td>(O5) User turns</td>
<td>How many user turns (help requests, hesitations)?</td>
</tr>
<tr>
<td><strong>Interaction Quality</strong></td>
<td></td>
</tr>
<tr>
<td>(O6) User help requests</td>
<td>How many user help requests?</td>
</tr>
<tr>
<td>(O7) User hesitations</td>
<td>How many user hesitations?</td>
</tr>
<tr>
<td>(O8) User false actions (all)</td>
<td>How many false user actions overall?</td>
</tr>
<tr>
<td>(O9) User false actions (navigation)</td>
<td>How many false user navigation actions?</td>
</tr>
<tr>
<td>(O10) User false actions (REs)</td>
<td>How many false user manipulations?</td>
</tr>
<tr>
<td><strong>Task Success</strong></td>
<td></td>
</tr>
<tr>
<td>(O11) Binary Task Success</td>
<td>Was the game won or lost?</td>
</tr>
<tr>
<td>(O12) Graded Task Success</td>
<td>Was the game won without problems, with small problems, with severe problems or lost?</td>
</tr>
</tbody>
</table>

longer than a pre-specified threshold of 4 seconds). Under the interaction quality category, we count the number of user help requests and user hesitations (the sum of which corresponds to the ‘user turns’ metric under interaction efficiency), the number of false user actions overall, the number of false user navigation actions, and the number of false user manipulation actions (i.e., false button presses). The ‘false user actions overall’ metric corresponds to the sum of false navigation and manipulation actions. Finally, under the category task success, we distinguish average binary task success (won or lost) from average graded task success (GTS) which penalises task difficulty in different ways, as defined in Section 5.3.1.

Binary task success is always 1 if a game was won (regardless of the number of problems) and 0 if it was lost. For graded task success, we assume that every user hesitation or help request indicates a problem, and assign the value of 2/3 (small problems) for more than five user turns, the value of 1/3 (severe problems) for more than ten user turns, and the value of 0 for a lost game. The objective metrics were designed based on PARADISE, but tailored specifically to our scenario, so as to measure the success of instructions in a situated interaction scenario. Results of the objective metrics were induced automatically from log files.

Table 5 shows the subjective metrics we use to evaluate the user satisfaction of our two systems. While questions Q1-Q6 are taken almost directly from PARADISE, questions Q7-Q10 were included to test some specifics of our situated NLG scenario. These metrics were obtained through questionnaires that participants were asked to fill after each game they played.
Table 5. Subjective metrics and the questions that were asked to obtain them.

<table>
<thead>
<tr>
<th>Subjective Metric</th>
<th>Question</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Q1) Easy to Understand</td>
<td>Was it easy to understand the system?</td>
</tr>
<tr>
<td>(Q2) Task Easy</td>
<td>Was it easy to find the trophy?</td>
</tr>
<tr>
<td>(Q3) Interaction Pace</td>
<td>Was the speed of the interaction okay?</td>
</tr>
<tr>
<td>(Q4) What to Do</td>
<td>Did you know at each moment what to do?</td>
</tr>
<tr>
<td>(Q5) Expected Behaviour</td>
<td>Did the system work as you expected it to?</td>
</tr>
<tr>
<td>(Q6) Future Use</td>
<td>Would you use this system in the future?</td>
</tr>
<tr>
<td>(Q7) Appropriate Help</td>
<td>Did the system help you appropriately when you needed it?</td>
</tr>
<tr>
<td>(Q8) Enjoy Game</td>
<td>Did you enjoy the game?</td>
</tr>
<tr>
<td>(Q9) Recommend to Friend</td>
<td>Would you recommend the system to a friend?</td>
</tr>
<tr>
<td>(Q10) Naturalness</td>
<td>Was the language of the system natural (non-robotic)?</td>
</tr>
</tbody>
</table>

6.2.2 Experimental setup

Setting and Participants We compare two systems for the GIVE task in a human evaluation study involving 19 participants, 79% (15 out of 19) female and 21% (4 out of 19) male, with an age average of 24.5.\(^9\) The two systems to be compared generated instructions for the GIVE task in three different worlds, which were chosen to be different from the training worlds, in order to assess the generalisability of our learnt policies. We thus used the hierarchy of policies that was trained in the training worlds and evaluated them in the evaluation worlds (rather than training a separate hierarchy of policies specifically for the evaluation worlds). The learnt NLG policy was therefore environment-independent. Future work can in addition investigate how policies can be adapted during interactions via online learning.

In the evaluation, one system used a jointly optimised policy, the other system used a policy that was optimised in isolation. Participants were asked to play three games. They were chosen so as to ensure that each participant played with at least one jointly optimised system and one system optimised in isolation. Apart from this condition, systems were chosen randomly from a uniform distribution.

Evaluation worlds For the human evaluation, we used the virtual worlds from the official GIVE challenge 2.5\(^10\) of 2011 (Striegnitz et al., 2011). They are shown in Figure 9. While the main skills required in the training worlds (cf. Figure 7) were navigation and disambiguation of a medium level of complexity, the evaluation worlds require a range of different skills. While evaluation world 1 was designed to

\(^9\) While we cannot exclude the possibility that the strong gender bias had an impact on our results, both GIVE challenges were faced with a similar situation. GIVE-2 had 79% of male participants, while GIVE-2.5 was slightly more balanced with 58%. Despite the gender bias found in both evaluations, no significant effect on task success or the subjective metrics was found in either evaluation.

\(^10\) \url{http://www.give-challenge.org/research/page.php?id=give-2.5-index} [accessed April 22, 2013]
Hierarchical Reinforcement Learning for Situated NLG

Fig. 9. Illustration of the evaluation worlds. World 1 is most similar to the training worlds in that it requires navigation and disambiguation skills at a medium level of difficulty. World 2 is focused on referring expression generation and World 3 on complex navigation.

be similar to the training worlds, evaluation world 2 focuses on referring expressions. A large number of same-coloured buttons are located close to each other in different spatial arrangements so that disambiguation becomes a challenge. Evaluation world 3 requires sophisticated navigation skills in all rooms, especially in a maze-like corridor in which users can quickly lose orientation or a room full of alarm tiles where any wrong step may cause the alarm to be triggered. Finally, it includes a room with many small rooms that require precise navigation.

6.2.3 Experimental results

Following the human evaluation study, we analysed the results in order to draw conclusions with respect to the effects that a joint or an isolated optimisation has on interactions and user satisfaction. Overall, the analysis is based on 57 games.

Objective metrics. Table 6 compares average results (with their corresponding standard deviations) for the joint and the isolated setting and shows the \( p \)-values indicating the significance of the comparison between both settings. We can see that the jointly optimised system performs better than the system that was optimised in isolation according to almost all metrics. It produces shorter interactions using fewer words and turns and causes fewer user turns and hesitations and higher task success. The key findings can be summarised as follows.

- The isolated policy produces significantly more system words (O3) than the joint policy (\( p < 0.04 \)). This difference could be interpreted as a suboptimal balance between \textit{efficiency} and \textit{detail} in instructions. When the joint policy is able to achieve an equal (or higher) task success using fewer words, the isolated policy most likely included redundant detail.
- The isolated policy produces significantly more system words per turn (O4) than the joint policy (\( p < 0.0001 \)). This difference again points to a suboptimal balance of choosing or organising utterance contents. The cognitive load that is imposed on the user during an interaction is increased with the number of system words per turn that the user needs to keep in mind. (Unnecessarily) long utterances can therefore lead to user confusions and affect task success.
Table 6. Results for the objective evaluation metrics per policy for the joint and the isolated setting. The objective metrics are organised into three groups, interaction efficiency (EFF) (the lower values, the better), interaction quality (QUA) (the lower values, the better) and task success (TS) (the higher values, the better). Numbers in the third and fourth column refer to averages (per game) and are given together with their standard deviations. The final column shows \( p \)-values for the comparison obtained with a paired t-test. The best result per metric is indicated in bold-face.

<table>
<thead>
<tr>
<th>Objective Metric</th>
<th>Joint</th>
<th>Isolated</th>
<th>( p )-val.</th>
</tr>
</thead>
<tbody>
<tr>
<td>(O1) Elapsed Time (mm:ss)</td>
<td><strong>11:40 ± 5.54</strong></td>
<td>13:19 ± 4.56</td>
<td>0.28</td>
</tr>
<tr>
<td>(O2) System turns</td>
<td><strong>314.5 ± 151.3</strong></td>
<td>330.2 ± 80.6</td>
<td>0.7</td>
</tr>
<tr>
<td>EFF</td>
<td><strong>3025.3 ± 1522</strong></td>
<td>3887.7 ± 1271</td>
<td><strong>0.04</strong></td>
</tr>
<tr>
<td>(O3) System words</td>
<td>9.6 ± 1.3</td>
<td>12.1 ± 1.5</td>
<td><strong>0.0001</strong></td>
</tr>
<tr>
<td>(O4) System words per turn</td>
<td>20.7 ± 14.0</td>
<td>22.2 ± 8.7</td>
<td>0.7</td>
</tr>
<tr>
<td>(O5) User turns</td>
<td>2.4 ± 3.0</td>
<td><strong>2.0 ± 1.4</strong></td>
<td>0.6</td>
</tr>
<tr>
<td>QUA</td>
<td>18.4 ± 12.6</td>
<td>20.3 ± 8.7</td>
<td>0.5</td>
</tr>
<tr>
<td>(O6) User help requests</td>
<td>21.4 ± 15.5</td>
<td><strong>19.3 ± 6.9</strong></td>
<td>0.6</td>
</tr>
<tr>
<td>(O7) User hesitations</td>
<td><strong>11.0 ± 7.6</strong></td>
<td>12.8 ± 6.2</td>
<td>0.4</td>
</tr>
<tr>
<td>(O8) User false (all)</td>
<td>10.3 ± 10.4</td>
<td><strong>6.5 ± 3.7</strong></td>
<td>0.1</td>
</tr>
<tr>
<td>(O9) User false (navigation)</td>
<td>0.80 ± 0.4</td>
<td>0.61 ± 0.5</td>
<td>0.1</td>
</tr>
<tr>
<td>(O10) User false (manipulation)</td>
<td><strong>0.43 ± 0.3</strong></td>
<td>0.23 ± 0.2</td>
<td><strong>0.009</strong></td>
</tr>
<tr>
<td>TS</td>
<td>0.61 ± 0.5</td>
<td>0.23 ± 0.2</td>
<td><strong>0.009</strong></td>
</tr>
</tbody>
</table>

- The joint policy achieves higher task success than the isolated policy. While the difference in terms of binary task success (O11) only shows a statistical trend (\( p < 0.1 \)), the difference in graded task success (O12) is significant at \( p < 0.0009 \). This means that users interacting with the joint policy encounter fewer problems and experience more smooth and successful interactions. This is also reflected in the large difference between binary and graded task success.

The comparison of the joint and the isolated policy seems to suggest that a joint optimisation leads to shorter, more efficient and more successful interactions. An exception to the overall trend is represented by metric O8, the number of false user actions overall, and metric O10, the number of false manipulation actions, i.e. wrong button presses. While users of the joint policy press on average 10.3 (±10.4) wrong buttons, users of the isolated policy press only 6.5 (±3.7) wrong buttons on average. The reason for this is most likely that few users in the joint setting pressed a very high number of wrong buttons, as is indicated by the high standard deviation of the O10 metric. The majority of users pressed very few (or no) wrong buttons, however.

Subjective metrics The subjective user ratings indicate the user satisfaction with each system. Table 7 summarises the results, where the last column in the table provides the \( p \)-value for the comparison of the previous two columns. Overall, we
Table 7. User satisfaction results per policy (scores range from 1 to 5 and are the better, the higher). Numbers refer to averages per game and are shown with standard deviations. The last column shows \(p\)-values for the comparison of systems. The best results per metric are indicated in bold-face.

<table>
<thead>
<tr>
<th>Subjective Metric</th>
<th>Joint</th>
<th>Isolated</th>
<th>(p)-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Q1) Easy to Understand</td>
<td>3.4 ± 1.0</td>
<td>3.26 ± 1.09</td>
<td>0.6</td>
</tr>
<tr>
<td>(Q2) Task Easy</td>
<td>3.01 ± 1.02</td>
<td>3.0 ± 1.16</td>
<td>0.9</td>
</tr>
<tr>
<td>(Q3) Interaction Pace</td>
<td>2.9 ± 1.32</td>
<td>2.86 ± 1.45</td>
<td>0.9</td>
</tr>
<tr>
<td>(Q4) What to Do</td>
<td>3.43 ± 1.02</td>
<td>2.91 ± 1.08</td>
<td>0.08</td>
</tr>
<tr>
<td>(Q5) Expected Behaviour</td>
<td>3.67 ± 1.14</td>
<td>3.52 ± 1.03</td>
<td>0.6</td>
</tr>
<tr>
<td>(Q6) Future Use</td>
<td>2.6 ± 0.8</td>
<td>2.56 ± 0.89</td>
<td>0.9</td>
</tr>
<tr>
<td>(Q7) Appropriate Help</td>
<td>3.3 ± 1.04</td>
<td>2.87 ± 1.21</td>
<td>0.1</td>
</tr>
<tr>
<td>(Q8) Enjoy Game</td>
<td>2.95 ± 1.08</td>
<td>2.56 ± 1.03</td>
<td>0.1</td>
</tr>
<tr>
<td>(Q9) Recommend to Friend</td>
<td>3.0 ± 1.16</td>
<td>2.56 ± 1.26</td>
<td>0.1</td>
</tr>
<tr>
<td>(Q10) Naturalness</td>
<td>3.36 ± 0.95</td>
<td>3.21 ± 1.07</td>
<td>0.6</td>
</tr>
<tr>
<td><strong>Sum (maximal score 50)</strong></td>
<td><strong>31.62</strong></td>
<td><strong>29.31</strong></td>
<td></td>
</tr>
</tbody>
</table>

can see a clear tendency of users preferring the joint over the isolated policy. The user satisfaction ratings for all games can be summarised as follows.

- Users consistently rate the joint policy better than the isolated policy, even though unfortunately none of the differences is statistically significant.
- The metric ‘Expected Behaviour’ (Q5) receives the highest ratings for both the joint (3.67 ± 1.14) and the isolated (3.52 ± 1.03) policy. In turn, ‘Future Use’ (Q6) receives the lowest, 2.6 (±0.8) for the joint policy and 2.56 (±0.89) for the isolated policy. For the latter case, the metrics ‘Enjoy Game’ (Q8) and ‘Recommend to Friend’ (Q9) are rated similarly low. Especially the metrics Q8 and Q9 can mean that users of the isolated policy enjoyed their games less than users of the joint policy. The metric ‘Future Use’ in contrast could also have a different interpretation. Users may not have seen the usefulness of using the game in the future because they are not interested in video games: on a scale of 1 (i.e. ‘playing never’) to 5 (i.e. ‘playing very often’), our participants rated themselves as playing video games between ‘rarely’ and ‘never’ (1.438). An alternative interpretation is that users found the pace of the interaction too fast, as indicated by the ‘Interaction Pace’ (Q3) metric, so that slowing the interaction pace down could lead to higher user satisfaction.
- The metric ‘What To Do’ (Q4) showed the biggest difference in user ratings between the joint (3.43 ± 1.02) and the isolated (21 ± 1.08) system. While it is not statistically significant, it shows the strongest trend among all individual subjective categories. Users seemed to find instructions generated by the joint system more easy to interpret and felt more safely guided through the task.

Despite an overall trend that users seem to prefer the joint over the isolated policy, we were not able to report any significant differences. Related work on the
evaluation of spoken dialogue systems suggests a factor analysis (Möller et al., 2007; Wolters et al., 2009; Dzikovska et al., 2001). An explanatory factor analysis explains the variability found in a set of observed, correlated variables in terms of a set of unknown latent variables, or factors. These factors are often fewer than the initial set of variables and reveal those underlying subjective categories that users were concerned about in their ratings. The advantage of a factor analysis is often that it reveals those subjective experiences with a system that matter to users, rather than reflecting the system designer’s expectations—as is often the case with pre-defined questionnaires. Please see Hone and Graham (2000) for details on a factor analysis applied to spoken language processing. A factor analysis applied to our subjective metrics of the GIVE evaluation showed the following. An illustration is provided in Figure 6.2.3.

Two factors were identified as accounting for 65% of the variability found in user ratings. For Factor 1, which we can call \textit{Usability}, subjective metrics (Q4) ‘What to Do’, (Q8) ‘Enjoy Game’ and (Q9) ‘Recommend to Friend’ had high factor loadings of $>0.80$. Factor loadings indicate the correlations between questionnaire items. For Factor 2, which we can call \textit{Pace}, only subjective metric (Q3) ‘Interaction Pace’ had a high factor loading of $>0.80$. The difference between the joint and the isolated policy for factor \textit{Pace} was not significant at $0.9$. While the difference for factor \textit{Usability} was not significant either, at $p<0.07$, at least, we can observe a statistical trend for this factor. All in all, these results indicate that statistical significance may have been achieved here if more data was available.

\section*{6.3 Comparison with Systems from the GIVE Challenge}

To allow for a comparison of our hierarchical RL framework with other state-of-the-art approaches to situated NLG, Table 8 contrasts our results with objective and subjective metrics collected for several systems in the GIVE-2 and GIVE-2.5 challenges. The former was run in 2010 and collected games from 1825 participants. The latter was run in 2011 and collected 536 games. The official results were discussed in Koller et al. (2010) and Striegnitz et al. (2011), respectively. GIVE-2.5 was run with the same evaluation worlds as our evaluation. The worlds in GIVE-2 were comparable in that all three worlds posed different challenges for the systems. World 1 was designed to be most similar to the training worlds, while World 2 focused on referring expressions and World 3 on navigation. All evaluations were therefore carried out in comparable, if not identical, virtual worlds. All subjective scores in the table were rescaled from the $-100$ to $+100$ scale used in GIVE to our 1 to 5 scale.

We chose seven systems for our comparison, the two best systems of GIVE-2 (\textbf{NA} and \textbf{S}) and the five best systems from GIVE-2.5 (\textbf{P1}, \textbf{P2}, \textbf{C}, \textbf{CL} and \textbf{L}). Since the overall results of GIVE-2.5 were better than of GIVE-2, we included more systems from the latter challenge in order to make a more challenging comparison.\footnote{No systems from GIVE-1 are compared because the setup of the first study was different} There is unfortunately not always a perfect match between subjective metrics,
but we wanted to include them nevertheless for a more comprehensive point of comparison. In particular, not all questions that we asked participants were the same that GIVE participants were asked. For category Q3, while we asked subjects *Was the speed of the interaction okay?*, GIVE asked participants to rate the statement *The system’s instructions were visible long enough for me to read them*. For category Q4, we asked *Did you know at each moment what to do?* while the GIVE questionnaire contained *I was confused about which direction to go in*. Finally, while we asked *Did the system give you appropriate help when you needed it?* for category Q7, GIVE used *The system immediately offered help when I was in trouble*. All objective and other subjective categories have a direct correspondence. Unfortunately, the number of questionnaire items differed in GIVE-2 and GIVE-2.5, so that some fields in the table cannot be compared. Since we are comparing data from separate evaluations, the results in Table 8 serve more as an indication rather than a direct

in that users were not able to move through the environment freely, but had to move in discrete steps.
Table 8. Objective and subjective metrics for our systems (J=Joint and I=Isolated) compared with the best systems of the GIVE-2 challenge (NA and S) and the GIVE-2.5 challenge (P1, P2, L, C, CL). * indicates measures taken from Benotti and Denis (2011b) rather than the GIVE challenge.

comparison and statistical significance is not reported. We can nevertheless make a number of observations from the data comparison:

- In terms of task success, we can see that our joint policy outperforms all other systems by at least 10%. This result also holds for other GIVE systems which were published separately from the challenge, such as Garoufi and Koller (2010) who achieve 69% and Benotti and Denis (2011b) who achieve 70%. This result reflects our reward function which placed a substantial weight on task success, rather than other metrics such as instruction or interaction length.

- The other objective metrics seem to suggest that both of our systems generate significantly more instructions and are more verbose than the other GIVE systems, which led to longer interaction times. This reflects the generation strategy learnt by our system, which was able to combine high-level and low-level instructions and aggregate several instructions into one. This produced many instructions such as Go left and then towards the blue button. In contrast, many GIVE systems relied predominantly on shorter instructions such as turn left or press blue.

- In terms of the subjective metrics, we can see that our system is slightly outperformed in Easy to Understand and Interaction Pace. The latter was already indicated in our own evaluation, where participants wished that in-
Instructions were displayed slightly longer and the system would reduce its overall interaction speed. On the other hand, our system performs substantially better in What to Do than most competitors and was ranked in the middle for Appropriate Help.

- We can further see that participants considered our system’s instructions more natural than its competitors’, enjoyed playing more and would recommend the game to a friend more often. In terms of naturalness, this is again reflected in our reward function, where we placed an explicit weight on human-like surface forms. To an extent, the other metrics confirm our earlier results in that participants enjoy playing when they win the trophy and they do not enjoy playing when they lose. Participants may therefore have enjoyed playing with our system most because it achieved the highest task success score overall.

- Finally, we can see that while the isolated policy is outperformed in many categories, it is still able to compete with some systems, such as in the categories Interaction Pace, Enjoy Game, Recommend to Friend, Naturalness and Binary Task Success. This indicates that even a policy optimised in isolation represents a competitive baseline.

The highest overall scores in this comparison were achieved by two rule-based systems, C (Racca, Benotti, and Duboue, 2011) and L (Denis, 2011). This suggests that a carefully designed ad hoc solution to a problem can still outperform many data-driven systems in NLG nowadays. Systems P1/P2 (Garoufi and Koller, 2011b) and CL (Benotti and Denis, 2011a) represent more state-of-the-art approaches. System P1 was using a combination of planning and supervised learning to NLG that aimed to maximise the understandability of referring expressions (P2 acted as a planning-only baseline). This system received good scores for Interaction Pace and Appropriate Help, possibly because its planning steps guided users in small steps avoiding confusions and maximising understandability. System CL used a corpus-based selection approach, choosing instructions from a pre-collected corpus of human utterances in the same domain. This system was rated well for Naturalness. The reason is probably that it relied on instructions that humans produced for the very same situation the system was facing. On the other hand, this method does not take context into account which can lead to inconsistencies and low scores in other subjective categories. In summary, the comparison with these systems shows that our hierarchical RL approach is able to achieve comparable performance to state-of-the-art systems: while our joint policy is outperformed in some subjective categories, it achieves higher task success and more enjoyable and natural interactions than the other systems. This corresponds to the optimisation metrics that our reward function was designed for.

7 Conclusions and future directions

Natural Language Generation systems for interactive contexts are faced with numerous trade-offs in generating an utterance that is optimally adaptive to the user.
and situation. Trade-offs include the level of detail chosen in a situation as well as the speed and efficiency with which instructions can be generated within a dynamic and constantly changing context. This article has suggested to address these challenges using hierarchical reinforcement learning. It extends previous research on NLG for interactive systems in several ways. First, it represents a novel hierarchical optimisation framework for situated NLG. This model is based on a divide-and-conquer approach and optimises a hierarchy of subtasks rather than one single complex task. In this way, it is more scalable for large state-action spaces than previous approaches towards reinforcement learning for NLG. Second, this hierarchical model has been trained with a comprehensive data-driven reward function addressing several aspects of our situated scenario. In contrast, related work has focused either on hand-crafting reward functions or has induced them for single aspects of the task only. Finally, we have compared two different learning settings for our domain, a joint setting in which a policy is learnt with pre-defined shared knowledge across subtasks, and an isolated setting without any shared knowledge. Results from simulation and a task-based human evaluation study showed the benefits of the joint architecture in optimising the trade-off between efficiency and detail in situated interaction. The joint setting led to more successful and efficient interactions that were better perceived by human users than their isolated counterpart.

Some future research directions are summarised in the following.

First, the idea of jointly optimising the behaviour of distinct, but related, subtasks is likely to enhance the performance of systems beyond NLG and dialogue. Candidate areas for such a joint treatment are language analysis and production, or multi-modal systems, where a joint treatment could help to reinforce communicated contents with non-linguistic behaviours.

Second, RL agents typically learn a behaviour policy offline during a training phase in a simulated environment and then execute the learnt policy statistically during deployment. To allow agents to learn from real interactions, however, via online learning and adaptation, more efficient training algorithms are needed that allow action values to be computed quickly and reliably so that they could immediately have an impact on the agent’s current behaviour. See Bohus, Langner, Raux, Black, Eskenazi and Rudnicky (2006), Cuayahuitl and Dethlefs (2011a) and Gašić, Jurčiček, Thomson, Yu and Young (2011) for some first advances.

Third, RL agents are typically designed by a system developer who bases his or her design decisions on knowledge of the task, the domain or the end user of the system. Drawbacks are that system development can be slow and labour-intensive, and different design decisions can have different effects on the performance of a system. An interesting direction for future research is therefore the investigation of methods for inducing the structure and features of the learning agent automatically from human or domain data. In this way, hierarchy construction could be automated to accelerate development times and increase reuse of resources. Simultaneously, the benefits of a modular architecture and of using a divide-and-conquer approach would be preserved for easy maintenance and scalability to large search spaces.
Automatic feature induction is also interesting for deciding which features should be \textit{shared} between agents for a joint optimisation.

Fourth, RL agents for NLG currently make the simplifying assumption that their knowledge about the user and environment is complete. This assumption is often unrealistic because most environments are not fully observable. While research on Partially-Observable environments has been done on dialogue systems (Williams and Young, 2007), \textit{generation under uncertainty} has yet to be transferred to research on trainable NLG.

Fifth, our model relies on tabular state representations which can affect its scalability as the state-action space grows. While we have suggested a hierarchical setting to address this problem, \textit{function approximation} techniques, such as linear approximation, neural networks or decision trees, are an alternative (or complementary) method to enhance scalability. Some approaches for dialogue include Henderson, Lemon and Georgila (2008), Jurcícek, Thomson and Young (2011), Pietquin, Geist, Chandramohan and Frezza-Buet (2011) and Cuayáhuítl, Kruijff-Korboyová and Dethlefs (2012).

Finally, to evaluate our suggested methods on a larger scale, we would like to \textit{transfer} hierarchical RL to \textit{new domains}, such as text generation, and \textit{new applications}, such as sentence compression, summarisation or machine translation.

\textbf{Appendix A: Hierarchical State and Action Space}

This section provides a detailed description of the knowledge and actions available to each learning agent in the hierarchy in Figure 5, on page 14. Each agent will be shown as a feature structure and explained in turn. In the state representation $S^i_j$, variables shown in cursive fonts are \textit{shared} variables. This means that they were originally state variables of single agents which can now be accessed by other agents as well. This is in order to take them into account for their own decision making. All non-cursive variables are individual state variables that cannot be accessed by other agents. This is the main difference between learning a jointly optimised policy and a policy optimised in isolation: while in the former, agents can access shared state variables, in the latter only individual, non-shared variables exist. The state space of the isolated agent can therefore be obtained by excluding all cursive state variables. In the action set $A^i_j$, bold-face actions denote composite actions, and the goal state $G^i_j$ defines the termination conditions for the agent.
Agent $M_0^0$, first of all, is the root agent which initiates all generation episodes. It can either choose a primitive action such as to confirm a previous user action, *Well done!*, tell the user to stop navigating, *Wait!*, or not to press a button, *Not this one!*. Alternatively, it can choose a composite action and pass control down to a child subtask. Agent $M_1^1$ is responsible for references and agent $M_1^2$ is responsible for navigation instructions.

Specifically, agent $M_0^1$ deals with generating references to buttons or landmarks. It can make decisions based on the visibility of the next goal, the presence of landmarks and the reference context. It should also make sure that an utterance plan has been chosen before presentation to the user. If a button reference needs to be generated, it may, e.g., call child subtask $M_2^0$.
Hierarchical Reinforcement Learning for Situated NLG

\[ S_2 \]

\[
\begin{align*}
& v_{24}: \text{ColourDistractor} \leftarrow \{0=\text{unfilled}, 1=\text{filled}\} \\
& v_{25}: \text{ColourReferent} \leftarrow \{0=\text{unfilled}, 1=\text{filled}\} \\
& v_{26}: \text{DiscriminatingColourDistractor} \leftarrow \{0=\text{false}, 1=\text{true}\} \\
& v_{27}: \text{DiscriminatingColourReferent} \leftarrow \{0=\text{false}, 1=\text{true}\} \\
& v_{28}: \text{Distractor} \leftarrow \{0=\text{unfilled}, 1=\text{filled}\} \\
& v_{29}: \text{HorizontalRow} \leftarrow \{0=\text{false}, 1=\text{true}\} \\
& v_{30}: \text{Horizontal} \leftarrow \{0=\text{unfilled}, 1=\text{filled}\} \\
& v_{31}: \text{NumberOfDistractors} \leftarrow \{0=\text{none}, 1=\text{one}, 2=\text{two or more}\} \\
& v_{32}: \text{PositionInConfiguration} \leftarrow \{0=\text{corner}, 1=\text{edge}, 2=\text{middle}, 3=\text{only button}, 4=\text{other}\} \\
& v_{33}: \text{Position} \leftarrow \{0=\text{unfilled}, 1=\text{filled}\} \\
& v_{34}: \text{Surface} \leftarrow \{0=\text{unfilled}, 1=\text{filled}\} \\
& v_{35}: \text{Type} \leftarrow \{0=\text{unfilled}, 1=\text{filled}\} \\
& v_{36}: \text{VerticalRow} \leftarrow \{0=\text{false}, 1=\text{true}\} \\
& v_{37}: \text{Vertical} \leftarrow \{0=\text{unfilled}, 1=\text{filled}\}
\end{align*}
\]

Agent \( M_2 \) also shows that many actions are complementary to each other. This means that there is an action pair, such as \textit{include distractor} and \textit{do not include distractor}, one of which needs to be chosen at each instance in order to update the corresponding state variable, here \textit{Distractor}, from \textit{unfilled} to \textit{filled}. This is a precondition for reaching the terminal state and ensures that all actions are considered by the agent. Since the reward function penalises the agent for each action it takes, it may otherwise happen that the agent neglects favourable actions in order to avoid a negative reward.

\[ A_2 \]

\[
\begin{align*}
& \text{referring expression } M_3^2, \\
& \text{include_distractor, do not include_distractor,} \\
& \text{include_type, do not include_type,} \\
& \text{include_referent_colour, do not include_referent_colour,} \\
& \text{include_distractor_colour, do not include_distractor_colour,} \\
& \text{include_horizontal_position, do not include_horizontal_position,} \\
& \text{include_position_in_configuration} \\
& \text{include_position_in_configuration}
\end{align*}
\]

\[ G_2 \]

\[
\begin{align*}
& \{ v_{24}=1, v_{25}=1, v_{28}=1, v_{30}=1, v_{33}=1, v_{34}=1, v_{35}=1, v_{37}=1 \}
\end{align*}
\]

In terms of navigation, agent \( M_1 \) is responsible for choosing a navigation level. It can choose low-level navigation by calling agent \( M_2 \) or high-level navigation by calling agent \( M_4 \). Mixed strategies can be generated by alternating these two.
choices. It can also decide to repair a previous navigation instruction (by calling agent $M_2^2$), in case the user was not able to comprehend it, and it should make sure that an utterance plan has been chosen before presentation to the user. Agent $M_1^1$ shares state variables on the aggregation and presentation strategy with the utterance planning agent $M_2^1$ so that a good balance between cognitive load and efficiency can be found.

The child agents of task $M_1^1$, low- and high-level navigation, both deal with content selection of their particular navigation type. Agent $M_2^3$ generates instructions of types direction, orientation or straight. It can optionally also include a destination or path instruction. Agent $M_2^4$ generates instructions of types destination and path, and optionally a referring expression, in case a button is a destination instruction. Both agents should ensure that the navigation instructions receive a surface realisation before being presented to a user by calling agents $M_3^1$,...,$M_3^5$.

Whenever an utterance plan is needed, agent $M_2^1$ can be called. This agent decides whether to aggregate a set of messages or not, and if so, whether to conjoin them or order them sequentially. It further chooses an information structure (whether the theme should be marked or unmarked) and possible temporal markers (first, second, then, now, etc.). Finally, it decides whether to present information in a composite manner, i.e. all in one, or incrementally, in a piece-meal fashion. The former usually supports efficiency, whereas the latter reduces cognitive load. The agent has access to the navigation level chosen in its state representation so that this can further be considered for choosing an appropriate presentation strategy.
Sometimes an utterance can be unsuccessful because the user was not able to comprehend or interpret it correctly. In such cases, agent $M_2^2$ may be called for a repair. It can either paraphrase a previously unsuccessful utterance, repeat it, or switch the current navigation strategy (from high to low level, e.g.).

Agents $M_3^0, ..., 5$, finally, deal with generating different surface realisations for the semantics of referring expressions (agent $M_3^0$), destination instructions (agent $M_3^1$), direction instructions (agent $M_3^2$), orientation instructions (agent $M_3^3$), path instructions (agent $M_3^4$), and instructions to go straight (agent $M_3^5$).
All surface realisation agents share knowledge of previous choices. During learning, each surface realisation agent will make its own surface realisation decisions and then evaluate their suitability under different circumstances given the rewards they yielded in different contexts (as described in Section 5.3.2).

Appendix B: An example generation episode

The feature structure on page 39 below shows an example generation episode for the utterance *Go to the sofa*, where control is passed from the root agent down to the bottom of the hierarchy. At all levels, decisions are made towards the final form of the utterance. Information is passed between agents in the form of state updates in the agent’s knowledge base.

Generation starts with the root agent \( M_0 \), whose state \( s_0 \) includes information on the current goal status and next system action required. Both kinds of information are stored in the agent’s knowledge base but originate from the virtual world. The agent then executes a navigation instruction, which is a composite action, so that control is passed to the child agent \( M_1 \). This is a content selection (CS) subtask. The current state here contains information on that the next goal is visible and on the current navigation level, which can be low, high or mixed. This information is available from the generation history. The agent also knows that the current route leg is short, that the user is not confused and has executed the previous instruction successfully. All this information enters the agent’s knowledge base through the virtual world (even though information about the user always originates in the user’s knowledge base, as shown in Figure 4). Finally, the agent has information on that the utterance planning agent has not yet been consulted, which is required though before execution can terminate. It therefore chooses to call agent \( M_1 \) next to perform utterance planning (UP). Control is passed to the new subagent, which again has to choose the best action given its current knowledge.

Note that agents \( M_1 \) and \( M_2 \) share several state variables, which are shown in cursive fonts. Agent \( M_1 \) has knowledge of the presentation type chosen, even though it has no control over the decision. It is an utterance planning decision and is therefore made in agent \( M_1 \). Similarly, agent \( M_2 \) knows the current navigation level, which is originally determined by agent \( M_1 \). Such shared state variables are the main mechanism through which a joint optimisation occurs. Since agents share knowledge with other agents, they are able to consider this knowledge in their own decision making process. For example, had agent \( M_1 \) seen that the presentation strategy chosen by \( M_2 \) was complex, it could have chosen a simpler navigation strategy. Or conversely, agent \( M_2 \) is able to take the high navigation level chosen by \( M_1 \) into account for choosing a presentation strategy that eases the user’s cognitive load. All other variables, which are not shown in cursive fonts, are specific to one agent and cannot be accessed by others.
Example generation episode for the utterance "Go to the sofa." Control is passed from parent to child agents whenever a composite action is invoked and is passed back upon termination. Agent names along with their subtask are shown in red, as are terminal states. (T) denotes the terminal state for agent $M_{ij}$.

Every time an agent takes an action, this is reflected in the updated state representation at the next time step.\footnote{For brevity, we show only the updated state variables and omit those that are unchanged from the original state.} Once utterance planning is complete with decisions on presentation style and temporal markers, control is passed back to the calling parent agent, $M_1$.

Subsequently, decisions are made to obtain a high-level navigation instruction (by calling agent $M^2_2$) and to obtain a surface form for a destination instruction (agent $M^3_1$, called by $M^2_2$). The latter is a surface realisation (SR) task, which bases decisions on lexical and grammatical information such as the verb, preposition and relatum to be realised. Once a surface form has been chosen, control is passed back to the calling agent and the destination slot is updated from *unfilled* to *filled*. At this point, control is passed back through several agents that have reached their terminal state and the generation process is completed. See Figure 6 for an illustration of the hierarchical state-action sequence of this example. For illustration, this example has relied on a subset of possible states and actions per agent.
Hierarchical Reinforcement Learning for Situated NLG

References


Cuayáhuitl, Heriberto and Nina Dethlefs. 2011b. Spatially-aware Dialogue Control Using Hierarchical Reinforcement Learning. ACM Transactions on Speech and Language Processing (Special Issue on Machine Learning for Robust and Adaptive Spoken Dialogue System), 7(3).


Dietterich, Thomas G. 2000. An Overview of MAXQ Hierarchical Reinforcement Learning. In Symposium on Abstraction, Reformulation, and Approximation (SARA), HorseShoe-Bay, Texas, USA.


