Learning the Parameters of Reinforcement Learning from Data for Adaptive Spoken Dialogue Systems

Layla El Asri

UNIVERSITÉ DE LORRAINE

École doctorale IAEM Lorraine

Learning the Parameters of Reinforcement Learning from Data for Adaptive Spoken Dialogue Systems

THÈSE

présentée et soutenue publiquement le 21 janvier 2016

pour l'obtention du

Doctorat de l'Université de Lorraine

(mention informatique)

par

Layla El Asri

Composition du jury

Président :	Fabrice Lefèvre
Rapporteurs :	Wolfgang Minker Fredéric Béchet
Examinateurs :	Christophe Cerisara Fabrice Lefèvre

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503
UMI 2958 (CNRS-GeorgiaTech) - Orange Labs

Mis en page avec la classe thesul.

"And again he thought the thought we already know: Human life occurs only once, and the reason we cannot determine which of our decisions are good and which bad is that in a given situation we can make only one decision; we are not granted a second, third, or fourth life in which to compare various decisions. [...] *Einmal ist keinmal. What happens but once might as well not have happened at all.* [...] *History is as light as individual human life, unbearably light, light as a feather, as dust swirling into the air, as whatever will no longer exist tomorrow."*

Milan Kundera. ***The unbearable lightness of being.***

Résumé

Cette thèse s'inscrit dans le cadre de la recherche sur les systèmes de dialogue. Ce document propose d'apprendre le comportement d'un système à partir d'un ensemble de dialogues annotés. Le système apprend un comportement optimal via l'apprentissage par renforcement. Nous montrons qu'il n'est pas nécessaire de définir une représentation de l'espace d'état ni une fonction de récompense. En effet, ces deux paramètres peuvent être appris à partir du corpus de dialogues annotés. Nous montrons qu'il est possible pour un développeur de systèmes de dialogue d'optimiser la gestion du dialogue en définissant seulement la logique du dialogue ainsi qu'un critère à maximiser (par exemple, la satisfaction utilisateur).

La première étape de la méthodologie que nous proposons consiste à prendre en compte un certain nombre de paramètres de dialogue afin de construire une représentation de l'espace d'état permettant d'optimiser le critère spécifié par le développeur. Par exemple, si le critère choisi est la satisfaction utilisateur, il est alors important d'inclure dans la représentation des paramètres tels que la durée du dialogue et le score de confiance de la reconnaissance vocale. L'espace d'état est modélisé par une mémoire sparse distribuée. Notre modèle, *Genetic Sparse Distributed Memory for Reinforcement Learning* (GSDMRL), permet de prendre en compte de nombreux paramètres de dialogue et de sélectionner ceux qui sont importants pour l'apprentissage par évolution génétique. L'espace d'état résultant ainsi que le comportement appris par le système sont aisément interprétables.

Dans un second temps, les dialogues annotés servent à apprendre une fonction de récompense qui apprend au système à optimiser le critère donné par le développeur. A cet effet, nous proposons deux algorithmes, *reward shaping* et *distance minimisation*. Ces deux méthodes interprètent le critère à optimiser comme étant la récompense globale pour chaque dialogue. Nous comparons ces deux fonctions sur un ensemble de dialogues simulés et nous montrons que l'apprentissage est plus rapide avec ces fonctions qu'en utilisant directement le critère comme récompense finale.

Nous avons développé un système de dialogue dédié à la prise de rendez-vous et nous avons collecté un corpus de dialogues annotés avec ce système. Ce corpus permet d'illustrer la capacité de mise à l'échelle de la représentation de l'espace d'état GSDMRL et constitue un bon exemple de système industriel sur lequel la méthodologie que nous proposons pourrait être appliquée.

Abstract

This document proposes to learn the behaviour of the dialogue manager of a spoken dialogue system from a set of rated dialogues. This learning is performed through reinforcement learning. Our method does not require the definition of a representation of the state space nor a reward function. These two high-level parameters are learnt from the corpus of rated dialogues. It is shown that the spoken dialogue designer can optimise dialogue management by simply defining the dialogue logic and a criterion to maximise (e.g user satisfaction).

The methodology suggested in this thesis first considers the dialogue parameters that are necessary to compute a representation of the state space relevant for the criterion to be maximized. For instance, if the chosen criterion is user satisfaction then it is important to account for parameters such as dialogue duration and the average speech recognition confidence score. The state space is represented as a sparse distributed memory. The Genetic Sparse Distributed Memory for Reinforcement Learning (GSDMRL) accommodates many dialogue parameters and selects the parameters which are the most important for learning through genetic evolution. The resulting state space and the policy learnt on it are easily interpretable by the system designer.

Secondly, the rated dialogues are used to learn a reward function which teaches the system to optimise the criterion. Two algorithms, reward shaping and distance minimisation are proposed to learn the reward function. These two algorithms consider the criterion to be the return for the entire dialogue. These functions are discussed and compared on simulated dialogues and it is shown that the resulting functions enable faster learning than using the criterion directly as the final reward.

A spoken dialogue system for appointment scheduling was designed during this thesis, based on previous systems, and a corpus of rated dialogues with this system were collected. This corpus illustrates the scaling capability of the state space representation and is a good example of an industrial spoken dialogue system upon which the methodology could be applied.

Acknowledgement

I would like to thank my advisors Prof. Olivier Pietquin and Dr. Romain Laroche for their priceless help during this project. Their advice, patience, and knowledge were precious in carrying out this work. Olivier has an extraordinary knowledge and understanding of the research in machine learning and he consistently pointed me to the right resources and explained the state of things in a very understandable manner. Romain has great technical skills and it often felt like he was understanding what I was doing faster and deeper than myself. They made a very complementary team and I am thankful for having had them as advisors. I want to also thank Prof. Minker, Béchet, Cerisara and Lefèvre for accepting to be part of the jury of my thesis. I would like to thank the NADIA team at Orange for their help on designing the appointment scheduling system and also for providing a very positive work atmosphere and following my project as it was being pursued. I am very thankful to our manager Rémi for his support and availability. At Orange, I would also like to thank the Innovation Primeur team for helping me with the data collection with the appointment scheduling system. The MALiS team at Metz and the CRIStAL lab at Lille always welcomed me warmly and I would like to thank everybody for interesting chats and insights. Of course, I have to thank other PhD students I have met during these years, Hatim for his humour and constant good mood, Senthil, Lucie, Bilal, Bassem, Vianney, and Edouard for making Metz fun, Hadrien for his kindness and his crazy ideas. Thanks a lot to Dr. Bilal Piot and Dr. Mathieu Geist for their contribution on the distance minimisation algorithm. On the personal level, thanks to my parents, my sisters, my sister-in-law and my brother for their unconditional love and support. A special thanks to my nephew Adam for being an endless source of joy and fun. I also want to thank all of my friends for being there and making me laugh so much, Arthur, Naima, Hajar, Assia, Fréjus, Christophe, and Nhat Minh.

Contents

List of Figures

List of Tables

Part I

Introduction

Wheatley: You two are going to love this big surprise. In fact, you might say that you're both going to love it...to death. Love it until it kil-until you're dead. [chuckles] All right? I don't know whether you're picking up on what I'm saying here, but...

GLaDOS (Genetic Lifeform and Disk Operating System): [interrupting] Yes, thanks. We get it. [Chell and GLaDOS enter an elevator] All right, he's not even trying to be subtle any more. Or maybe he still is, in which case: wow, that's kind of sad. [...] Either way, I'm getting the feeling he's trying to kill us.

Dialogue between two robots and a human (Chell) in the video game Portal 2.

"There seems no reason to believe that we are acquainted with other people's minds, seeing that these are not directly perceived, hence what we know about them is obtained through denoting." [Russell, 1905]

"The question is: are these words still the expressions of thoughts?" [Mercier, 2004]

Language[1] is the only tool that we have to let others be acquainted with our thoughts and feelings. Yet, philosophers like Brice Parain struggled with language because it lacks precision when it comes to expressing anything complex, because words are strange entities that control us rather than us controlling them. There is indeed **"backlash in the gears of language"**; if we try to say something, the meaning inferred by the person listening to us is likely to differ from the one intended [Sartre, 1990].

Parain concluded at some point of his research that orders were the only efficient way of communicating, that we could only be understood correctly if we specified a precise action for someone else to perform. Words could only make sense in a tightly ordered situation, with one giving the order and the other receiving, executing. Another way to see it is that one defines what the word means, how it relates to an action, and the other must take this specific definition.

Then, Parain's journey through language took him back to an existentialist conception of language. Words shape us as we pronounce them because every time we use a new word, we have to invent its meaning. Love, family, work, these are the daunting notions we have to define as we speak them.

Language is an imprecise tool that shapes us as we use it but still, it is our only way of knowing and being known. The problems in human-human communication are also found in human-machine dialogue, and to a greater extent in actuality. For technological reasons, it is even more difficult to be understood by a machine than by another human being. And like in human-human dialogue, it is often easier to use a task-oriented system and give a precise description of the task we would like to see accomplished than try to have complex thoughts understood as they were intended. Machines are still mostly engaging in action-based dialogues, to send emails or help us troubleshoot. Nevertheless, research in human-machine dialogue has been producing sustained effort and substantial progress.

[1] By *language*, we intend spoken and body language as well as any form of art.

Machines are now capable of understanding a broad vocabulary coming from different speakers, and parsing and accomplishing the descriptions of complex tasks. The industry of voice-enabled interfaces has been booming for the last few years and consequential investments are being made in research. The market of intelligent virtual agents is estimated to reach $2.2 billion in 2019, with a Compound Annual Growth Rate (CAGR) of 30% between 2014 and 2019. New needs for voice-enabled interfaces have also appeared with the recent emergence of the internet of things. We can hope that soon machines will also be capable of a sophisticated use of the language that shapes us, that is not only based on actions but is playful and ambiguous, to the point that they will be able to practice (or try to practice) subtlety and sarcasm like in the dialogue above.

In this thesis, the focus is on dialogue between humans and machines. If machines learn to speak elegantly, they will jointly need to learn to dialogue efficiently. We will be investigating ways to teach a machine to engage in a dialogue and use its language tools to converse efficiently with a human being.

Chapter 1

Context and Motivations

1.1 Overview

It is an exciting time for research in speech technologies. It has been a long way from the Interactive Voice Response (IVR) systems of the 90's to today's Spoken Dialogue Systems (SDS). IVR take as input voice and Dual-Tone Multi-Frequency (DTMF, using the telephone keypad). They are still popular to automate call centers because they allow to handle a large volume of calls and are quite efficient for simple tasks such as banking or contract management. However, IVR are mostly based on simple hand-written call flows and can handle only a limited amount of queries. Furthermore, this type of interaction has not been equally successful everywhere. Indeed, IVR are not very popular in Japan where the culture imposes strict rules of politeness and care. Recent dialogue systems, on the other hand, handle natural language and thus, can provide a higher level of interaction and services.

As said in the introductory section, most widely deployed commercial systems are focused on relatively simple action-based dialogues. These systems have encountered a very large success in the past few years and have become a popular feature in mobile devices, commercial web sites, and web browsers. The companies which make operating systems for mobile devices now all propose a vocal/text personal assistant: Siri for Apple, Google Now for Google, and Cortana for Microsoft. Other major industrial actors are Nuance with the Dragon mobile assistant and Samsung with S-Voice. These vocal personal assistants enable their users to perform multiple tasks via natural language, such as dictate and send emails, ask for the weather, or perform a search on the web. The tremendous advances that have been made on Automatic Speech Recognition (ASR) and Natural Language Understanding (NLU) with deep neural networks [Hinton et al., 2012] and large knowledge bases have assisted in making spoken and text-based dialogue systems more usable and in giving more freedom to vocal-assistant developers. Consequently, this year, Nuance and Viv, a company created by the team who made Siri announced a new generation of personal assistants, with a wider range of skills. For instance, it will be possible to teach new tasks to Nuance's assistant and Viv's conversational agent will be able to perform a multi-domain task.

However, the currently deployed dialogue systems only have limited dialogue capabilities. These agents most often try to minimize the duration of the dialogues they have with users and they model tasks as sets of slots to fill. For instance, if a user wants to send an email, Siri will ask for the following slots: recipient, object, and body of the email. And if the user asks for restaurants nearby, Siri will not try to have the query refined but instead it will return the complete list of restaurants it has found. Even in a purely action-based context such

as this one, introducing some dialogue is valuable. For instance, it allows to manage data presentation. The system could ask the user which type of restaurant she is looking for or what her budget is and then only display a few restaurants. Good dialogue management is also important to handle efficiently misunderstandings and improve system usability. Systems with sophisticated dialogue capabilities are yet to be deployed for several reasons. First, dialogue strategies must be defined. For instance, with the previous restaurant-seeking example, the system would need to ask for different criteria and then decide when to show results or if more criteria are needed. The system could also ask for many criteria at the same time or ask for them one by one. It would be tedious and non-transferable to hand-craft all the strategies for all the possible dialogues a system could have. Another point is robustness to ASR problems: the longer the interaction with a user, the more likely it is that their will be ASR misunderstandings. A dialogue system should come with a strategy to handle such misunderstandings.

To address these issues, research has successfully turned to Markov Decision Processes (MDP) and Reinforcement Learning (RL) [Levin et al., 1997, Lemon and Pietquin, 2007, 2012] for dialogue management. Modelling dialogue management as an RL agent means that the dialogue system is provided with the ability to learn how to interact with users. Instead of specifying the entire call flow in advance, the developer defines a set of actions that the system can take. Actions are for instance asking the user to provide or confirm a piece of information, or playing a help message. In order to decide what to say next, similarly to what a human would do, a dialogue system needs to keep track of the context of the dialogue. For instance, the system needs to keep track of the pieces of information the user has provided. Another example is that the system should play a help message only if it thinks that the user is having difficulty with the current interaction. This representation of the dialogue context is called the state of the system. RL prevents SDS designers from entirely hand-crafting the dialogue strategies of their systems. The designers only provide a set of states, a set of actions and a criterion to maximise (e.g., usability). The system then learns which action to perform at each state in order to maximise the criterion. RL also allows to build robust and scalable systems [Singh et al., 1999, Williams and Young, 2007].

An RL-based system chooses between different *actions* depending on its current *state*. For example, if the system's state indicates that the dialogue is not going well, the SDS could decide to play a help message or change its strategy to make the dialogue easier to follow for the user. The choice of action is based on *rewards* which are distributed to the system by a *reward function*. The reward function represents the task of the system. Ideally, it should be the most succinct, robust and transferable representation of the system's task [Russell, 1998]. For an SDS, the system's task is usually defined by a compound of business constraints such as minimising time on task and optimising user-centric metrics like usability [Lemon and Pietquin, 2012]. The most significant rewards are given at the end of the dialogue, when all of these metrics can be estimated. However, intermediate rewards can be given during the dialogue. For instance, it is possible to give a negative reward each time there is a user time-out, which means that the user did not say anything when she was supposed to [Laroche et al., 2011]. This is justified by the fact that if the user does not speak when she is expected to, it might be because she does not know what to say and the dialogue is hard to follow for this user. According to the states that the agent has encountered, the actions it has taken, and the rewards it has received, the RL agent computes for each state the action that will yield the highest expected sum of rewards [Sutton and Barto, 1998]. This mapping from states to actions is called a *policy*. An *optimal policy* maps each state to the action which is thought to maximise the expected sum of rewards that the system would receive if it started from that state and acted according to the same policy afterwards. The expected sum of rewards is optimised instead of just the immediate reward received after performing the action. This is explained by the fact that some rewards can be delayed, for instance the time on task is only known at the end of the dialogue. This

ability to deal with delayed rewards makes RL particularly well-suited for dialogue management.

Even though this framework has many advantages and answers all the reservations concerning dialogue systems, RL-based systems are not being widely deployed. However, previous endeavours should be mentioned. Indeed, in 2010, the NADIA (NAtural DIAlogue) team at Orange[1] deployed the first commercial SDS embedding RL [Laroche et al., 2010c]. In this work, Laroche et al. insisted on why RL-based dialogue systems had been difficult to put into production. The main reason is that industrial products are subject to the VUI-completeness principle [Pieraccini and Huerta, 2008] according to which "the behaviour of an application needs to be completely specified with respect to every possible situation that may arise during the interaction. No unpredictable user input should ever lead to unforeseeable behaviour". With RL, VUI-completeness is not guaranteed since the system's behaviour evolves as it tries out different strategies in order to learn the most efficient one given its reward function. Laroche et al. [2009] proposed a novel RL framework to respect VUI-completeness in commercial SDS. This framework relies on an automaton which allows to control the dialogue logic on a high level. Inside of each node of the automaton, a decision can be made based on RL but the overall automaton structure guarantees global coherence of the dialogue and VUI-completeness. However, even with this industrial-friendly structure, another problem hinders the development of commercial RL-based SDS. RL is a complex machine learning technique and not many SDS developers are familiar with it and all its subtleties. Therefore, RL for dialogue systems is still not perfectly suitable for wide industry deployment. In order to answer these challenges, research on easing the implementation of adaptive SDS has been carried. This work and the contribution of this thesis to this line of research are presented in the following sections.

1.2 Motivations

In an effort to simplify the development of RL-based SDS, two complementary lines of research have been pursued. The first line of research strives to define cross-domain frameworks to enable transferring the knowledge acquired with an SDS (e.g., the policy learnt by the system) to another SDS or to facilitate comparing two systems [Walker et al., 1997b, Bohus and Rudnicky, 2005, Gašić et al., 2013]. The second thread designs techniques to learn from data the parameters of an RL-based system, that is to say the state space representation and the reward function [Toney et al., 2006, Paek and Chickering, 2005, Boularias et al., 2010a, Sugiyama et al., 2012, Ultes et al., 2012]. The work presented in this thesis intends to contribute to this second thread.

Hand-crafting the state space representation of an SDS raises several issues [Paek, 2006]. The states of the system contain the information about the dialogue context upon which the decisions of the system are based. To perform an efficient dialogue, the system needs to have a good representation of the dialogue context and know how the dialogue is going, which pieces of information the user has provided, what is the task that the user is trying to perform, etc. Therefore, a system's ability to perform the task it was designed to achieve depends heavily on the relevance of the state space representation. In the RL framework, this requirement is translated as follows. The task of the system is represented by the reward function. At each state, the system learns to maximise the expected sum of rewards. As a consequence, the states of the system should allow to accurately estimate the expected sums of rewards. The states are a collection of dialogue parameters, such as indicators of the quality of the exchange with the user, the status of the slots which are needed to fill to make a query,... However, many dialogue parameters are available and the ones which should be included in the state space are *a priori* unknown. For instance, the number of help requests from the user might be relevant or not. In addition, this type of features can take many values, say between 0 and 10. If a state was to be created for

[1] http://www.orange.com/

each combination of values, for instance {# help requests = 0, # questions asked by the system = 2,...}, the size of the state space would explode and the system would need a cumbersome number of dialogues to accurately estimate the expected sum of rewards at each state. To answer these difficulties, research has turned to learning a compact representation of the state space from data [Paek and Chickering, 2005, Toney et al., 2006, Rieser and Lemon, 2011, Gašić et al., 2013]. Nonetheless, the previously proposed techniques are either lacking the ability to deal with many features or an easily interpretable result. An interpretable policy is crucial because if the developer can easily understand the behaviour of the system, she can use this knowledge in order to improve best practices and design more efficient dialogue systems in the future.

The reward function should also be designed carefully and hand-crafting it quickly leads to several limitations. Deciding when and how to distribute rewards to the system are crucial choices. However, it is not clear how to combine different desiderata into one reward. For instance, it is not obvious to decide if the system should be penalised with a negative reward each time there is a user time-out or how dialogue duration should be included in the rewards. If for instance the system's task is to maximise usability, it is fair to think that shorter dialogues should be preferred. Yet, it is not obvious whether dialogue duration should count more than user time outs. Hand-crafting the rewards in order to optimise interaction quality is thus a very difficult task for the SDS designer, despite her intuition of how it will influence the system's behaviour. Besides, a bad design of this function can have catastrophic consequences on the system's learning. As a natural answer to this obstacle to RL-based SDS modelling, research on adaptive SDS has again explored the possibility to learn the rewards from data instead of hand-crafting it according to intuition [Walker, 2000, Boularias et al., 2010a, Sugiyama et al., 2012]. In this work, two algorithms are proposed. They both learn a reward function from a set of dialogues. These functions are designed to maximise the dialogue ratings. The contribution of these algorithms compared to previous work is that the reward functions can be used for online learning and that they are entirely based on the dialogue ratings. The advantages are that once again, the results are easy to interpret for the designer and that learning speed is better than if an estimation of the dialogue rating was only given at the end of the dialogue.

Despite these previous efforts, all in all, there is a lack of global methodology to automatically learn a policy from a rated corpus, without having to define neither the state space nor the reward function. There are disparate algorithms to either learn the reward function or a representation of the state space but there is no general consistent framework which would enable more industrial actors to put into production RL-based systems with extended dialogue capabilities. This is what this thesis intends to provide. The work presented in this document was carried among the NADIA team at Orange, UMI 2958 at CNRS-GeorgiaTech, and CRIStAL lab (UMR 9189) at CNRS/Lille 1/ECL. The purpose was to propose a framework which would respect the industrial constraints announced earlier. From this context, three guidelines for this work have emerged:

- First, it is supposed that the designer is not familiar with RL. The work asked to the designer only concerns the dialogue logic and the actions the SDS can perform. This first guideline stipulates that the reward function and the representation of the state space should be automatically learnt from data.

- Even if the developer is not familiar with RL, it is important to provide her feedback and insight on what the system learns. The second guideline is thus that the result of learning should be easily interpretable by the designer.

- Finally, the third guideline is to build models that can be used for batch and online learning. This is important because while the parameters are learnt from data, the system's behaviour should keep

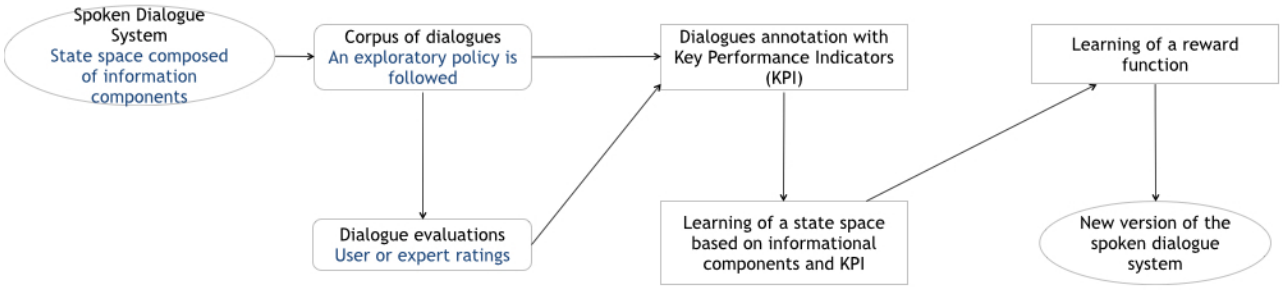


Figure 1.1: Methodology to automatically learn a policy from a set of rated dialogues.

improving with new dialogues and potentially different user behaviours. The design methodology as well as the contributions of this work are described in the next section.

1.3 Contributions

The contributions of this thesis are the following:

- A methodology to learn a policy from a set of rated dialogues, without having to define the reward function nor the state space.
- An algorithm that learns an interpretable and memory-efficient state space.
- Two algorithms that learn a reward function which aims to speed up learning.
- An SDS to schedule appointments and a corpus of rated and annotated dialogues with this system.

The methodology is proposed in Figure 1.1. At first, the designer is asked to provide an SDS with a state space only composed of *informational components*. These components represent the system and the user's common context and intentions [Larsson and Traum, 2000]. For instance, with the email example, an informational component might be the state of knowledge concerning the recipient: is it unknown, known, or confirmed? This first state space gives a general view of the logic of a dialogue with the system: the designer characterises the dialogue states that should be encountered by the system and the possible actions at each of these states. For instance, the designer might give a set of possible actions for when the user has informed the recipient. These actions could be: asking for a confirmation of the email address or asking for the object of the email.

Then, it is proposed to optimise this first hand-crafted version of the interface with RL. For this purpose, data is required, more precisely, dialogues where a pure exploratory policy is set. A pure exploratory policy is one that tries the different actions at a state in a uniform fashion. This is required because learning the state space and the reward function necessitates as many observations as possible at each state in order to be reliable.

The designer is then asked to provide a criterion to optimise. This could simply be user satisfaction but it could also include business-oriented parameters like dialogue duration. In Figure 1.1, the criterion is a single value provided by the user who did the call or by an external expert who rated the call after listening to it. Thus, after this step, the developer has a corpus of rated dialogues where many different policies were followed.

The dialogues in the corpus are then annotated. This step is done automatically from dialogue logs. Dialogue features are extracted from the logs and added to the informational components. Let us consider for example that the current informational component is that the user has provided an email address for the recipient. Key Performance Indicators (KPI) such as the dialogue duration so far or the number of user time-outs that have occurred so far are added to this information. At each step of a dialogue, the information known comes both from informational components and KPI. The chosen KPI have been shown to be related to quality ratings given by users or experts [Walker et al., 1997a, 2002, Evanini et al., 2008].

Based on this augmented information, a representation of the state space is computed. The solution proposed in this thesis allows to include a high number of continuous or discrete KPI (dialogue duration in seconds, number of help requests, number of user time-outs,...) in the state space. Our solution automatically selects the features which are relevant for learning. Another important aspect of this solution is that it returns a representation which makes the learnt policy easily interpretable. This is crucial because if the developer can easily understand the behaviour of the system, she can use this knowledge in order to improve best practices and design more efficient dialogue systems in the future. As said in the previous section, previously proposed techniques were either lacking the ability to deal with many features or an easily interpretable result.

The next step is to compute a reward function on the resulting state space. In this work, two algorithms are proposed. They both learn a reward function from a set of dialogues. The functions are designed to teach the system to maximise the ratings. The main contributions of these algorithms compared to previous work is that the reward functions which are returned can be used for online learning and that they speed up learning compared to the case when an estimation of the rating is only distributed at the end of the dialogue. A policy is then learnt on the corpus, with the new state space representation and reward function.

To summarise, from a corpus of rated dialogues, the main parameters of reinforcement learning are learnt as well as a policy. Then, the system can be deployed with an optimised behaviour. The designer can therefore focus on the dialogue capabilities of the system without having to hand-craft end-to-end dialogues.

We propose to apply this methodology on appointment scheduling systems. The appointment scheduling task is chosen because it requires negotiation between the user and the SDS in order to decide on a common time slot. It is a good example of when dialogue is absolutely necessary to accomplish a task. Two SDS were designed. First, NASTIA (Negotiating Appointment SeTting InterfAce, [El Asri et al., 2014d]) which is based on previous work [Laroche et al., 2011]. Its design follows the industry-fit framework developed at Orange. Then, a simulator for the appointment scheduling task is also used to generate dialogues to compare the different algorithms. These two systems illustrate two settings. In the case of NASTIA, the constraints of the system are simple: NASTIA has a list of available appointments and can book any of these appointments without any preference. Accepting an appointment is therefore a binary decision based only on the system's availabilities and is not learnt with RL. The system's task only consists of optimizing ratings related to user satisfaction. The simulator, on the other hand, has different preferences for its available slots. This system needs to learn to optimize the preference of the appointment slot along with ratings related to user satisfaction. The simulator will be used to validate the algorithms and methodology whereas NASTIA will serve to illustrate the state space representation on real data and showcase its interpretability. NASTIA will also be used to train the user satisfaction estimator for the simulator.

1.4 Outline

Part I gives the elements of background which are necessary to understand the work carried out during this thesis, namely SDS design, RL for SDS, and SDS evaluation. Several experimental choices ensue from this study, including the appointment scheduling systems' dialogue strategies and the protocol followed for dialogue collection with NASTIA.

Part II defines the dialogue systems and RL tasks which are used for experimental purposes. Chapter 4 lists the already existing SDS and RL problems which are chosen to experiment on different aspects of the reward and state space inferring algorithms. Then, Chapter 5 describes the appointment scheduling systems and the corpus of dialogues collected with NASTIA. The KPI chosen for the state space are also presented in this chapter. This part illustrates the first three steps of the methodology on real data.

The next step of the methodology is tackled in Part III which is dedicated to the representation of the state space. The first chapter of this part, Chapter 6 specifies the problem and reviews previously proposed solutions. Based on this, a new solution is proposed in Chapter 7. This solution allows to handle the many KPI included in the state space and it returns a compact and interpretable representation. In the same chapter, this solution is evaluated on the simulator and another RL problem and it is shown to outperform a comparable approach in terms of learning speed and compactness.

The last step of the methodology is the inference of a reward function from ratings. This question is studied in Part IV. In Chapter 8, the problem is stated and previous solutions are discussed. A first algorithm is proposed in Chapter 9. First, estimators of dialogue performance are discussed because this algorithm requires one. It is shown on a corpus of rated dialogues that Support Vector Ordinal Regression (SVOR) outperforms other estimators on many metrics. Then the reward inferring algorithm using SVOR is described and tested on the appointment scheduling simulator. Afterwards, Chapter 10 presents a second algorithm which does not rely on an estimator. The reward functions computed by these two algorithms can be used for online learning and are shown to increase learning speed compared to the function which gives the dialogue as a reward at the end of the dialogue.

Finally, Chapter 11 discusses the results of this thesis and proposes directions for future work.

List of Publications

Layla El Asri, Romain Laroche, and Olivier Pietquin. Reward function learning for dialogue management. In *Proc. of STAIRS*, 2012.

Layla El Asri, Romain Laroche, and Olivier Pietquin. Reward shaping for statistical optimisation of dialogue management. In *Proc. of SLSP*, 2013.

Layla El Asri, Hatim Khouzaimi, Romain Laroche, and Olivier Pietquin. Ordinal Regression for Interaction Quality Prediction. In *Proc. of ICASSP*, 2014a.

Layla El Asri, Romain Laroche, and Olivier Pietquin. Régression Ordinale pour la Prédiction de la Qualité d'Intéraction. In *Proc. of JEP*, 2014b.

Layla El Asri, Romain Laroche, and Olivier Pietquin. Task completion transfer learning for reward inference. In *Proc. of MLIS*, 2014c.

Layla El Asri, Romain Laroche, and Olivier Pietquin. DINASTI: Dialogues with a Negotiating Appointment Setting Interface. In *Proc. of LREC* , 2014d.

Layla El Asri, Rémi Lemonnier, Romain Laroche, Olivier Pietquin, and Hatim Khouzaimi. NASTIA: Negotiating Appointment Setting Interface. In *Proc. of LREC*, 2014e.

Layla El Asri, Romain Laroche, and Olivier Pietquin. Compact and interpretable dialogue state representation with genetic sparse distributed memory. In *Proc. of IWSDS*, 2016a.

Layla El Asri, Bilal Piot, Mathieu Geist, Romain Laroche, and Olivier Pietquin. Score-based inverse reinforcement learning. In *Proc. of AAMAS*, 2016b.

Chapter 2

Designing a Spoken Dialogue System

This chapter provides background on SDS, RL, and SDS evaluation. First, the architecture of SDS is detailed and then the focus is set on dialogue management. Afterwards, the discussion is on RL for SDS and the different frameworks which were proposed in the past. These first two sections provide the necessary background to understand how the appointment scheduling SDS were designed. The last section gives insight on how to collect dialogues with an SDS and which kinds of ratings can be asked from users or experts. This section also explains the data collection process with NASTIA. In the rest of this document, when notations are not specified, they refer to the definitions of this chapter.

2.1 Spoken Dialogue Systems

2.1.1 Definition

During a *dialogue*, two or more interlocutors exchange information. An SDS is a conversational interface which permits dialogue between a machine and a human-being, in natural language. An SDS receives an input in spoken natural language and produces a vocal output in natural language too.

Several types of dialogue systems have been implemented. A type of SDS which has been predominant are *task-oriented* systems. They are built in order to assist a user in the achievement of a goal. For instance, one can say that Siri is task-oriented in that it helps the user perform several tasks and its purely conversational abilities are very limited[1]. Examples of academic task-oriented SDS are *NJFun* [Litman et al., 2000], which provides information about things to do in New Jersey and *Let's Go* [Raux et al., 2003], which gives bus schedules and itineraries in Pittsburgh.

Recently, academic research has also deployed non task-oriented *conversational* systems. These interfaces are not to accomplish a specific task but are built in order to have a conversation with the user about a given theme. For instance, Higashinaka et al. [2008] researched social interaction with an SDS having a conversation about the animals that are liked or disliked by the user. In the same vein, Meguro et al. [2009] worked on a *listening-oriented* system: the system listens to the user and provides feedback in order to make the user feel like she is being listened to.

Applications of SDS are multifaceted [Pietquin, 2004] and it is not intended here to be exhaustive. Widely-spread systems are *form-filling* systems which collect values for a predefined set of fields in a form [Danieli

[1] Siri can tell a joke or two but cannot engage in a general non task-oriented conversation.

and Gerbino, 1995, Litman and Pan, 1999, Lemon et al., 2006]. For instance, email-sending is a form-filling task and its fields are the object, the body, and the recipient of an email. Other applications of SDS are for instance *troubleshooting* [Acomb et al., 2007, Laroche et al., 2010b] and *tutoring* [Litman and Silliman, 2004, Daubigney et al., 2013].

2.1.2 History of Spoken Dialogue System Research at Orange

In the 70's, McCarthy and Hayes proposed to embed mental qualities (such as beliefs and intentions) into dialogue systems [McCarthy and Hayes, 1969, McCarthy, 1979]. The underlying idea was to describe the internal state and the behaviour of the system in a human-like manner. Following this idea, Newell proposed to build dialogue systems on the basis of beliefs and goals [Newell, 1980]. After that, many other studies proposed to add other mental qualities [Bratman, 1987, Cohen and Levesque, 1990, Sadek, 1991, Konolige and Pollack, 1993].

In particular, at Orange, the BDI (Belief-Desire-Intentions [Bratman et al., 1988]) system was used to build an interaction theory [Sadek, 1991] which was implemented [Bretier, 1995] into a dialogue system called ARTIMIS [Sadek et al., 1997, Sadek, 1999]. ARTIMIS and its underlying logic system were improved during almost a decade with complex action plans [Louis, 2002], a preference-based operator [Meyer, 2006], and uncertainty handling [Laroche et al., 2008].

Despite the strong theoretical results and the richness of the formalism, the logical approach has been abandoned for two main reasons: first, it could not guarantee VUI-completeness [Pieraccini and Huerta, 2005] and second, it was too complex for a non logic-expert to design a dialogue application on her own.

Therefore, since 2007, Orange has been relying on an industrial solution called Disserto. This solution enables to build SDS as automata, similarly to what most of the contemporaneous speech technology actors (Speechcycle, Nuance, AT&T) do. Disserto has been proved to be simple enough for programmers, solid enough for commercial dialogue systems, and flexible enough to support state-of-the-art research.

With Disserto, and under the impulse of the CLASSiC European project, many research projects have been pursued and published. The first subject of research focused on the reconciliation of dialogue research in industry and academy [Paek, 2007]. Following the seminal work of Singh et al. [2002] and Williams [2008], a hybrid reinforcement learning model called MVDP [Laroche et al., 2009, Laroche, 2010, Laroche et al., 2010c] (see Section 2.2.5) was designed and implemented into appointment scheduling systems. These systems were the first commercial dialogue application embedding online reinforcement learning [Laroche et al., 2010c] and they outperformed state-of-the-art systems from academic research during the CLASSiC final evaluation [Laroche et al., 2011]. In the same vein, an effort was made in order to display reinforcement learning results onto the design view [Laroche et al., 2010a]. This context was the basis upon which our research project was launched.

Afterwards, during the period of this thesis, an interest for *incremental dialogue processing* [Schlangen and Skantze, 2011] has appeared. Incremental dialogue makes dialogue systems more efficient [Aist et al., 2007, Paetzel et al., 2015] and human-like [Edlund et al., 2008]. A methodology to transform a traditional dialogue system's architecture into an incremental one has been proposed [Khouzaimi et al., 2014a], and reinforcement learning has been used to optimise the handcrafted strategy. This has been shown to give even better results [Hatim Khouzaimi and Lefèvre, 2015].

Disserto and the wide were a strong help to imagine and implement the methodology proposed in this dissertation. NASTIA, for instance, was designed with Disserto and included incremental strategies.

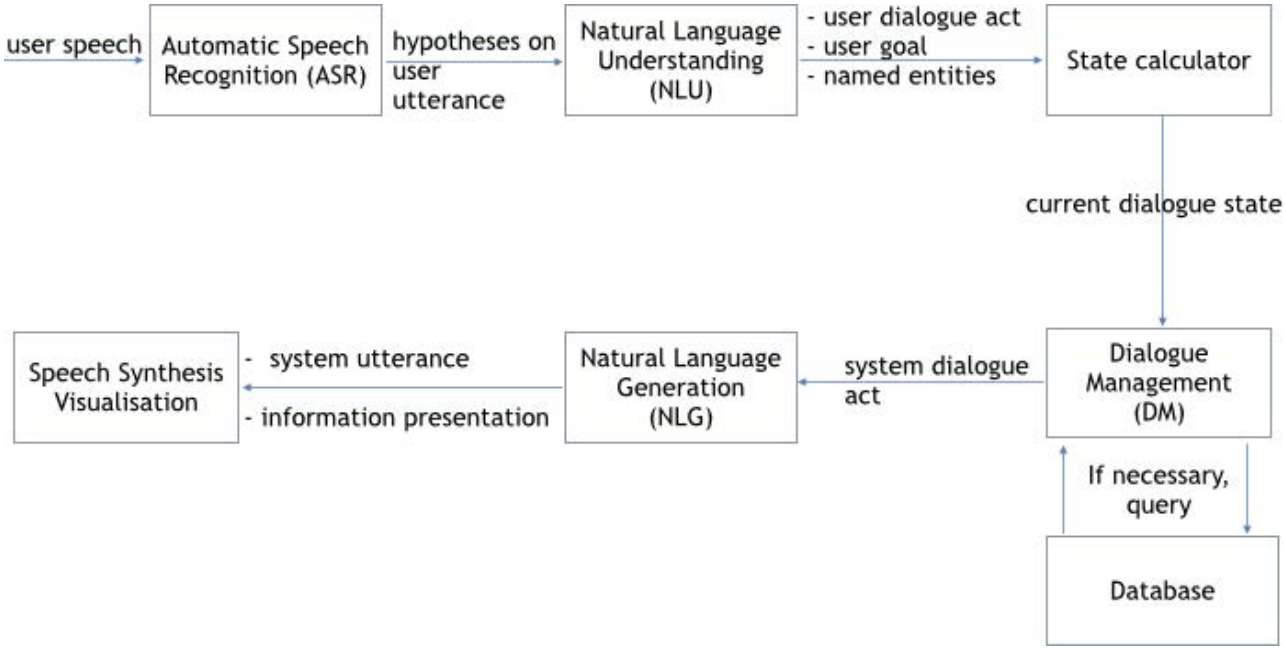


Figure 2.1: Overview of the different components of a spoken dialogue system.

2.1.3 Architecture

To build an SDS, the general architecture illustrated in Figure 2.1 is commonly adopted. Let us walk through this schema module by module according to what happens when a user speaks to the system. Let us consider that the user said "I want to book an appointment on Friday morning". The user's vocal input is treated by the ASR module which formulates a set of hypotheses on what was said. For instance: "I want to book an appointment on Friday morning" and "I want to look an appointment on Friday morning". Each hypothesis is associated with a confidence score. This score is a measure of how confident the ASR is concerning the recognition of the user's utterance. The list of hypotheses associated with the n highest confidence scores, returned in decreasing order, is called the n-best list of hypotheses. This list is then transmitted to the Natural Language Understanding (NLU) module which translates each hypothetical utterance into a User Dialogue Act (UDA) to provide to the state calculator. A dialogue act represents the meaning of an utterance at the level of the speaker's intention of producing this utterance [Austin, 1962]. The dialogue act encodes the type of utterance, for instance REQUEST, CONFIRM, as well as domain-related information. For example, the UDA corresponding to "I want to book an appointment on Friday morning" could be REQUEST_APPOINTMENT. In all that follows, dialogue acts will be written with upper-case letters.

Based on the latest UDA, the state calculator updates the dialogue state. This state might include dialogue history and hypotheses on the user's goal in addition to the list of UDAs returned by the NLU module. For instance, the state could be <UDA = REQUEST_APPOINTMENT, user_goal = {this week, Friday, morning}>. According to the state returned by the state calculator, the DM[2] chooses the next System Dialogue Act (SDA)

[2]The abbreviation *DM* will be used indifferently to refer to Dialogue Management or Dialogue Manager, as long as the context does not leave any ambiguity on the intended meaning.

and it might also query a database in order to answer the user's request. Still following the same example, the DM would query the database to see if any appointment is available on Friday morning and the SDA returned would depend on the result. It could be to inform that there is no appointment available on Friday morning or to ask the user for a preferred time. The SDA is then passed to the Natural Language Generation (NLG) module which forms the system's utterance: "There is not any availability on Friday morning". Finally, the utterance is transformed into vocal output by the speech synthesis module. In the case of a multi-modal system, the output and input can be composed of both vocal signal and actions on a graphical interface. The visualisation and information presentation items on Figure 2.1 concern these systems. In the following sections, the modules of SDS are further described. Then the definition and functions of the dialogue manager will be further discussed for it is the focus of this thesis.

Automatic Speech Recognition

The role of the ASR module in an SDS is to process the vocal input from the user and return a list of hypotheses on what was said: to a speech input X are matched word sequences $W_1, ..., W_n$. This task brings in many challenges. Many major challenges were first tackled from the 1970's to the early 90's at Carnegie Mellon University (CMU), IBM, and AT&T Bell Laboratories. These challenges were about handling continuous speech output (CMU's Hearsay-I system in 1973), performing unrestricted vocabulary dictation (CMU's Dragon system in 1974), and building a speaker-independent interface (CMU's Sphinx-I system in 1987) [Huang et al., 2014]. The Word Error Rate (WER) is one of the standard metrics to assess speech recognition systems. The string of words recognised by ASR is aligned to the one spoken by the user with dynamic string alignment. The number of words that were added, deleted and substituted is counted. The WER is then computed by dividing this number of erroneous words with the number of words in the user's utterance [Jurafsky and Martin, 2000]. This metric evaluates the transcription accuracy. Recent research on machine learning, in particular deep learning with recurrent networks, has enabled to reduce dramatically the WER of speech recognition systems [Hinton et al., 2012, Hannun et al., 2014].

For each input X, the ASR returns a list of n pairs of hypotheses and confidence scores. The hypotheses are the word sequences which match best the user input. The confidence scores are real numbers between 0 and 1 or 0 and 100 measuring how confident the ASR is about each hypothesis. The n-best list of hypothesis-confidence score pairs is passed to the NLU module whose role is to extract a meaning for each hypothesis. Commonly, a threshold is set on the ASR confidence score: if the highest confidence score is below the *ASR threshold*, all the hypotheses are rejected. The ASR threshold is set in order to find the best trade-off between false rejections and acceptances.

Natural Language Understanding

The NLU task is twofold: an NLU module has to define the meaning of a sentence and give a representation for this meaning. Since there is no general consensus on what an arbitrary sentence should mean, the NLU of an SDS is often task-dependent. The NLU then simply extracts task-specific arguments in each sentence. The representation of the utterance can thereby be framed in a simple way [Macherey, 2009]. Concept representation is commonly used for NLU: the meaning of a sentence is concisely divided into a set of concept values, a concept being the smallest chunk of meaning being relevant to the task [Levin and Pieraccini, 1995]. The concept representation for email sending could be $<$ recipient = Jean Dupont, object = Tomorrow's meeting $>$. NLU can also infer other characteristics about the user's gender, the user's level of confidence in what was just

said (*e.g.* "I know" *vs.* "I think"), etc. depending on what is important to perform the task of the SDS. Along with the understood concepts, the NLU returns a UDA, such as SEND_EMAIL. The concepts (and possibly other characteristics such as gender,...) returned by the NLU module can be viewed as the arguments of the UDA: { SEND_EMAIL, < recipient = Jean Dupont, object = Tomorrow's meeting >}.

Once NLU has been performed and the system has a set of hypotheses on the user's utterance, the dialogue manager decides the system's next action.

State Calculator and Dialogue Management

The DM is responsible for the dialogue's progress. Traum and Larsson [2003] give the following definition for DM. The DM is in charge of: updating the dialogue context, communicating with other back-end systems or databases to coordinate the system's next move, and deciding what to say next to the user and when to say it. In this thesis, the dialogue management task is defined as proposed by Traum and Larsson, except that this task is split between the state calculator and the dialogue manager: updating the dialogue context is performed by the state calculator, and the other functions are performed by the DM. The next move chosen by the DM is passed to NLG.

Natural Language Generation

NLG has a prominent role in the naturalness perceived by an SDS user. For a matter of convenience and cost reduction, it is common to rely on templates for NLG: the NLG module possesses a set of templates that it has to fill with values collected during the dialogue. For instance, " What is the object of the email you want to send to <recipient>?" could be a template. Nevertheless, building an adaptive NLG is interesting, for instance, to adapt the system's vocabulary to the user's. Repeating the user's vocabulary (priming effect) or aligning to the user's technical level (especially in a troubleshooting context) makes the interaction easier to follow for the user. Therefore, NLG models combining rule-based and stochastic models have been proposed [Galley et al., 2001].

Speech Synthesis

The final module of the dialogue chain performs speech synthesis. Static template prompts that do not need any data collection (*e.g.* welcoming prompts, fixed help messages...) can be recorded in advance. The other prompts are generated on the fly. The synthesis can include various meta-characteristics concerning for example prosody and pitch to express emotion.

2.1.4 Dialogue Management

State Representation

The *Information State* (IS) approach [Traum and Larsson, 2003] is a model for the DM task which has had a large success among SDS research. At each step of the dialogue, the state of an IS-based DM is described by informational components (beliefs, user model,...), formal representations of these components, dialogue moves that can trigger an update of the information state, update rules and an update strategy. The IS approach is a principled framework to implement a DM. Applications of this theory are numerous [Bos et al., 2003, Lemon et al., 2006, Roque et al., 2006, Young et al., 2009]. An IS might embed as many dialogue parameters as

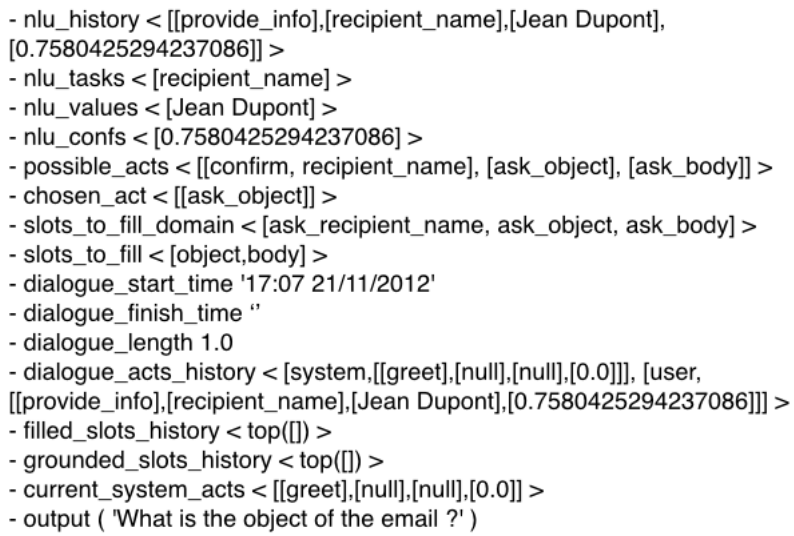


Figure 2.2: Possible representation of an information state for an email sending task.

the SDS designer judges to be relevant for the course of the interaction. An example of information-state based representation after the user has given the recipient of her email is given in Figure 2.2. The system keeps track of the history of the UDA understood by NLU (nlu_history) as well as the confidence scores corresponding to these UDA (nlu_confs). It also keeps track of filled and grounded slots. A filled slot corresponds to a piece of information given by the user (for instance, "Jean Dupont") and a grounded slot is a filled slot which has been confirmed. Based on the latest UDA, the DM chooses an action (chosen_act) among the possible ones (possible_acts). In this case, it chooses to ask the user for the object of the email.

Dialogue Logic

As said in the introduction, to apply the methodology proposed in this work, the SDS designer is asked to provide the informational components of the system. We define the informational components as: the system dialogue acts, the expected user dialogue acts, and possible actions given a UDA. This encodes the logic of a dialogue. For instance, the designer will indicate that the possible actions when the user has provided a recipient are to ask for a confirmation, ask for the body of the email, or ask for the object of the email. Now that a practical model for dialogue management has been introduced, let us discuss the science of dialogue management itself, that is to say what actions may be performed by the DM and how to use spoken language technologies to help a user achieve a task. This part is the domain of expertise of the SDS designer. It is covered here to explain the choices made for NASTIA and the dialogue simulations presented in the following part of the thesis.

Type of ASR problem	Strategy	System Utterance
Potential Misunderstanding	Explicit Confirmation	Do you want to send an email to Jean Dupont? Please answer yes or no.
Potential Misunderstanding	Implicit Confirmation	Send an email to Jean Dupont... What is the object of the email?
Rejection	Notify the user	I did not understand what you said.
Rejection	Ask the user to repeat	Could you repeat what you said?
Rejection	Ask the user to rephrase	Could you rephrase what you said?
Rejection	Repeat	Who do you want to send an email to?
Rejection	Prompt a help message	You can say: I would like to send an email to David Dickens.

Table 2.1: Typical automatic speech recognition error handling strategies in dialogue management.

Dialogue Pace

Let us start with the rhythm of the dialogue, that is to say, how the system and the user take turns to communicate. The *turn-taking* rhythm of a traditional interaction with an SDS is similar to walkie-talkie communication: the system speaks and only once the system's utterance is over the user can speak, and vice versa. However a dialogue between two humans is much more interactive due to the frequency of the interruptions [Strombergsson et al., 2013]. Recent research has thus focused on *incremental dialogue management*: a new interaction framework that enables the user or the system to barge in at any moment to clarify a point or ask for more precision about something [Schlangen and Skantze, 2011, Selfridge et al., 2013, Khouzaimi et al., 2014a,b]. The work presented in this thesis is based on the traditional turn-taking paradigm but it could be adapted in the future in an incremental setting. Note that the turn-taking strategy not only involves the DM but also ASR and NLU.

Dialogue Initiative Strategies

The second fundamental aspect of dialogue management is the *degree of initiative*. Three types of initiative are usually proposed: *user initiative*, *system initiative*, and *mixed initiative*. System initiative means that the system directs the dialogue and asks the user for any piece of information it might need. On the other hand, when user initiative is chosen, the user directs the dialogue and asks the system whatever she needs. A mixed initiative is as follows: the system has overall control of the dialogue but the user can barge in any time to lead the dialogue in another direction. Another choice concerns *closed* vs. *open* questions. Closed questions are such as "Who is the recipient of the email?", "What is the object of the email?", they request one piece of information at a time. An open question would be like "What action would you like to perform?". It enables the user to provide one or many pieces of information at a time, for instance: "I would like to send an email.", "I would like to send an email to Jean Dupont.".

Dialogue Error Recovery Strategies

As said in the introduction, when an SDS engages in a dialogue with a user, it should be able to deal with ASR misunderstandings and ASR rejections (which happen when the highest ASR confidence score for the latest

user utterance is under the ASR threshold). One common way of handling ASR misunderstandings is to ask the user to confirm what was understood by the system. This enables the system not to act upon a misunderstood utterance. Two types of confirmations are possible: implicit and explicit. An *explicit confirmation* consists of asking the user a yes/no question about the understood utterance. For instance, after the user has provided the recipient Jean Dupont for her email, the system could ask her: "Do you want to send an email to Jean Dupont? Please answer yes or no.". On the other hand, if the systems opts for an *implicit confirmation*, it simply repeats what it has understood and then it moves on with the dialogue. For instance: "Send an email to Jean Dupont. What is the object of the email ?". In this case, if the system is correct, the dialogue moves on without the need for an interaction with the user. Otherwise, the user can barge in and tell the system it misrecognised something. For instance: "Send an email to Jean Durand." "No, Jean Dupont.". The implicit confirmation strategy should then theoretically lower the lengths of the dialogues. However, studies have shown that it is not always adapted and some users might find it confusing [Sturm et al., 1999]. Sturm et al. showed that it is preferable to use this strategy if the ASR confidence score is high and if there is only one piece of information to confirm at a time.

As for ASR rejections, several recovery strategies exist. It usually starts with the system notifying the user that she was not understood. The system can then (or directly) ask the user to repeat or rephrase. In a more sophisticated way, the system can repeat the user's utterance without the missing piece of information. Finally, a last strategy consists of prompting a help message to hint the user about how to communicate with the system in order to be understood well [Yankelovitch, 1996]. The ASR-error-handling strategies are summarized and illustrated in Table 2.1.

Cooperativity

Finally, another important aspect which has been studied for dialogue management is *cooperativity*. Cooperativity in human-human dialogue was theorised by Grice [1989] as follows: "[one should make a] conversational contribution such as is required, at the stage at which it occurs, by the accepted purpose or direction of the talk exchange in which one is engaged". Grice derived from this definition a group of four maxims that can fully characterise a cooperative behaviour. These are the maxims of: quantity (provide the sufficient and necessary amount of information in your contribution), quality (do not say what you believe to be false and do not make suppositions about what you do not know), relation (be relevant) and manner (be clear). These maxims were adapted to human-machine dialogue and grouped into a set of principles by Bernsen et al. [1996] and Dybkjaer et al. [1996]. Wizard-of-Oz experiments[3] enabled the authors to add three principles to Grice's maxims: partner asymmetry (provide information about how to interact with the system, what the system can do), background knowledge (adapt the system's behaviour to user expertise), and repair and clarification (initiate repair or clarification in the case of communication failure). The cooperativity maxims and, by extension, Bernsen et al.'s principles, are well-suited to task-oriented Spoken Dialogue Systems (SDS) as, according to Grice, they correspond to a maximal efficiency in the pursuit of a shared goal. Dybkjaer et al. [1996] tested the principles on scenario-based dialogues with a flight reservation system and came to the conclusion that almost all of the interaction problems that occurred during the tests could be imputed to violations of one or more of the principles, thus validating the approach of designing task-oriented systems respecting the principles.

[3] In a Wizard-of-Oz (WoZ) experiment, a human replaces the dialogue manager and decides the system's actions based on the noisy input from ASR and NLU.

Turn-taking, the degree of initiative, ASR error handling and cooperativity are the main aspects of dialogue management. They will be further illustrated in the chapter dedicated to the appointment scheduling systems designed during this project. In the following section, RL for dialogue management is presented. The choice of initiative used to be the result of advanced user studies [Gorin et al., 1997, Litman and Pan, 1999, Walker et al., 1998]. RL was first proposed for dialogue management partly in order to relieve SDS designers from re-iterating the same process for each new SDS [Singh et al., 1999]. The main advantages of RL-based SDS compared to hand-crafted SDS are that RL allows data-driven design, it computes an optimal behaviour for a given objective function, it provides the capability to generalize from visited to unseen states, and it reduces design costs [Lemon and Pietquin, 2007]. It also allows to build robust and scalable systems.

2.2 Reinforcement Learning for Dialogue Management

The first decisions which were optimised with RL were the ones that had been made so far based on experience: open vs. closed questions, and the confirmation timing [Singh et al., 1999]. However, the RL framework enables to investigate choices that might not only concern DM but also NLG and information presentation (via speech synthesis or screen display). For instance, decisions concerning the optimal way to present information to the user [Walker, 2000, Rieser and Lemon, 2011] or the type of prosody to prompt the user with [Bretier et al., 2010] were investigated. Recently, RL has been applied to negotiating SDS where the goal of the system might differ from the one of the user [Heeman, 2009, Georgila and Traum, 2011, Barlier et al., 2015]. The choices the DM is concerned with are then when to make a proposition and when to offer an argument in favour of a specific proposition [Georgila, 2013]. First, RL is formalised in general and then in the SDS context.

2.2.1 Reinforcement Learning

Sutton and Barto [1998] gave the following definition for RL: “Reinforcement learning is learning what to do-how to **map situations to actions**-so as to **maximize a numerical reward signal**. The learner is not told which actions to take [...] but instead must discover which actions yield the most reward by trying them”. An illustrative example is the one of the operant conditioning chamber (also known as *Skinner box*). This chamber contains two levers: A and B, and a light. Initially, the light is off. If A is pushed while the light is off, the light turns on. If A is pushed when the light is on, the light goes off. If lever B is pushed when the light is on, some food is delivered at a corner of the chamber and if B is pushed with no light on, nothing happens. This chamber has been used to study the mechanisms of learning of animals such as rats. This system can be described in the terms of Sutton and Barto’s definition. The **learner** is the rat. The **situations** are the states encountered by the learner, that is to say, the presence or absence of light. The **actions** are the decisions of the learner: pushing A or B. The **reward** is the food. The rat is not told which sequence of actions to perform, it learns it with the objective of maximising its food income. In this case, the rat should start by turning the light on by pushing lever A and then pushing lever B to have the food delivered. This sequence of actions is illustrated in Figure 2.3. The most widely used paradigm to formalise RL is the MDP framework, which is presented in the following section.

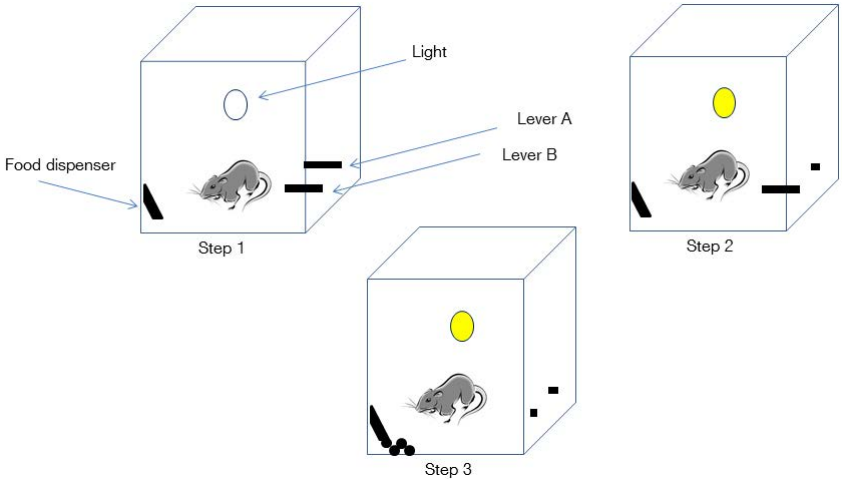


Figure 2.3: Optimal sequence of actions for the operant conditioning chamber.

2.2.2 Markov Decision Processes

Formalisation

The learner's progression through the environment is fractioned into time steps. In the previous conditioning chamber example, the time step t would be incremented after each action of the rat. Given a time measure and a time variable t, an MDP is defined by a tuple (S, A, P, R, γ) where:
- S is the *state space*. It represents the situations mentioned in Sutton and Barto's definition of RL. When there exists a set of final states $F \subset S$, the agent's task is said to be *episodic*.
- A is the *action space*. A *finite* MDP is an MDP with a finite state and action spaces.
- P is the *transition function*, which describes the environmental dynamics,

$$\begin{aligned} \forall a \in A, P_a\colon \quad & S \times S \to [0,1] \\ & (s, s') \mapsto P_a(s, s') = P(s_{t+1} = s' \mid s_t = s, a_t = a). \end{aligned}$$

- R is the *reward function*. Typically, it is defined on $S \times A$ or $S \times A \times S$. R_{t+1} is the immediate reward that the agent receives after having executed an action at time t. The reward function at (s, a, s') (resp. (s, a)) is the expected immediate reward for this transition (resp. state-action pair). Here, R is defined on $S \times A \times S$,

$$\begin{aligned} R\colon \quad & S \times A \times S \to \mathbb{R} \\ & (s, a, s') \mapsto R(s, a, s') = E[R_{t+1} \mid s_{t+1} = s', s_t = s, a_t = a]. \end{aligned}$$

- $\gamma \in [0, 1[$ is a *discount* (or deprecation) factor.

The Markov Property Assumption

An MDP is based on the *Markov property*, the transition probability to state s_{t+1} at time $t+1$ solely depends on the previous state s_t and the previous action a_t:

$$P(s_{t+1} \mid s_t, a_t, s_{t-1}, a_{t-1}, ..., s_0, a_0) = P(s_{t+1} \mid s_t, a_t).$$

In order to make this assumption hold, it is often necessary to include in the current state some information about the history of the state-action pairs previously visited by the agent [Pietquin, 2009].

2.2.3 Resolution

The conditioning chamber is a good illustration of the sequential aspect inherent to reinforcement learning. Indeed, the rat must perform a sequence of actions in order to get the food. An RL agent is not striving to optimise its reward at each time step but instead learns an optimal sequence of actions to optimise a global reward. This requires that the agent receives a long-term feedback for its actions. This is formalised by the *discounted cumulative reward*. The discounted cumulative reward received by the agent at time t is the discounted sum of the immediate rewards obtained starting from t:

$$r_t = \sum_{t' \geq 0} \gamma^{t'} R_{t+t'+1}. \tag{2.1}$$

The discount factor γ was introduced for continuous tasks (as opposed to episodic tasks), to avoid infinite returns. The adjustment of the value of the discount factor permits to give more or less importance to immediate rewards: the closer to 1 the factor, the more future rewards are taken into account and so, the more far-sighted the learning agent. A discount factor equal to 1 is only possible for an episodic RL task. In all that follows, *reward* will mean immediate reward and *return* will mean discounted cumulative reward.

The goal of the agent is to find a probability distribution on the state-action space $S \times A$, such that the return at each state is maximised. A mapping from $S \times A$ to a probability in $[0, 1]$ is called a *policy* π:

$$\begin{aligned} \pi : \quad & S \times A \to [0, 1] \\ & (s, a) \mapsto \pi(s, a) = P(a_t = a \mid s_t = s). \end{aligned}$$

A *deterministic policy* π maps a state to an action,

$$\begin{aligned} \pi : \quad & S \to A \\ & s \mapsto \pi(s) = a. \end{aligned}$$

Defining a deterministic policy is equivalent to assigning a probability of 1 to a state-action pair (s, a) and a probability of 0 to all the other actions. Given a policy π, the value $V^\pi(s)$ of a state s is the expected return starting from this state and following π,

$$V^\pi(s) = E[r_t \mid s_t = s, \pi]. \tag{2.2}$$

V is the *value function* or V-function and $V^\pi(s)$ is the V-value of s. Similarly, the value of a state-action pair (s, a) is

$$Q^\pi(s, a) = E[r_t \mid s_t = s, a_t = a, \pi]. \tag{2.3}$$

Q is the *state-action value function* or Q-function and $Q^\pi(s,a)$ is the Q-value of (s,a). It follows from these definitions and the Markov property that $\forall\, s \in S$,

$$\begin{aligned} V^\pi(s) &= E_\pi\left[r_t \mid s_t = s\right] \\ &= \sum_a \pi(s,a) \sum_{s'} P_a(s,s') \left[R_a(s,s') + \gamma E_\pi \left[\sum_{t'\geq 0} \gamma^{t'} R_{t+t'+2} \mid s_{t+1} = s' \right] \right] \\ &= \sum_a \pi(s,a) \sum_{s'} P_a(s,s') \left[R_a(s,s') + \gamma V^\pi(s') \right]. \end{aligned} \tag{2.4}$$

Similar computation leads to

$$Q^\pi(s,a) = \sum_{s',a'} P_a(s,s') \left[R(s,a,s') + \gamma \pi(s',a') Q^\pi(s',a') \right]. \tag{2.5}$$

These two equations are called the *Bellman equations*. They allow to express the value of a state (resp. state-action pair) as a function of the values of the other states (resp. other state-action pairs) of the MDP. The V-function and Q-function are the unique solutions to their Bellman equations.

Two policies can be compared on the basis of the V-functions under each policy: a policy π is better than or equivalent to a policy π' if the expected return for each state when following π is higher or equal to the one expected when π' is followed, that is to say if, $\forall\, s, V^\pi(s) \geq V^{\pi'}(s)$. The relation between policies is strict if at least one of these inequalities is strict. It is always possible to find at least one policy for an MDP that is better than any other one [Bertsekas and Tsitsiklis, 1996, Sutton and Barto, 1998]. Such policy is called *optimal* and noted π^*. The optimal V-function and Q-function corresponding to π^* are respectively noted $V^*(s) = \max_\pi V^\pi(s)$ and $Q^*(s,a) = \max_\pi Q^\pi(s,a)$. The *Bellman optimality equations* are then

$$V^*(s) = \max_{a \in A} \sum_{s'} P_a(s,s') \left[R(s,a,s') + \gamma V^*(s') \right], \tag{2.6}$$

$$Q^*(s,a) = \max_{a' \in A} \sum_{s'} P_a(s,s') \left[R(s,a,s') + \gamma Q^*(s',a') \right]. \tag{2.7}$$

When the transition probabilities and the expected rewards are known, these equations can be solved exactly with dynamic programming [Sutton and Barto, 1998]. However, most of the time, neither the transition probabilities nor the expected rewards are known in advance. *Model-based* algorithms solve the RL problem by presupposing that P and R are known or by building a model for them as the agent interacts with the environment [Sutton and Barto, 1998, Brafman and Tennenholtz, 2003, Strehl and Littman, 2005]. *Model-free* algorithms, on the other hand, suppose that these parameters are unknown throughout the agent's learning [Sutton and Barto, 1998, Watkins, 1989]. The RL algorithms used in this thesis will be presented along with their applications. For these algorithms to be efficient, it is important to tackle an important problem in RL known as the *exploitation-exploration trade-off*.

2.2.4 Exploitation-Exploration Trade-off

At each time step of RL, the agent must act in its environment based on its estimation of the action-value function. The agent might choose, at a state s, to perform the action a^* maximising the current Q-function:

$$a^* = \operatorname*{argmax}_a Q^\pi(s,a). \tag{2.8}$$

This behaviour is named *greedy*. This strategy relies on the current estimation of the Q-function. However, if this estimation is wrong, the agent might opt for a suboptimal action and ignore actions that might have led to better results. A greedy behaviour might thus lead to an under-optimal exploration of the state space. This is likely to happen at the beginning of learning, when knowledge of the space is still insufficient for the agent to be sure of the optimality of its decisions, hence the need for a trade-off between exploitation and exploration. The agent's exploration of the space should be sufficient to estimate correctly the values of the state-action pairs.

Several strategies have been proposed in order to find this trade-off. The simplest one is *ϵ-greedy* according to which the action a^* is chosen with probability $1 - \epsilon$ and a random action is chosen with probability ϵ. The parameter ϵ is a real number set at a small value, e.g., 0.1. More sophisticated exploration strategies have been proposed. For instance, *UCB1-TUNED* [Peter Auer and Fischer, 2002] minimises the regret caused by exploration. The regret is the loss of return entailed by performing a sub-optimal action. An action a is chosen according to $Q^\pi(s, a)$ but also according to the number of visits to (s, a). In practice, the algorithm computes a confidence interval for $Q^\pi(s, a)$ and the agent performs the action which has the highest Upper Confidence Bound (UCB).

RL for DM has been introduced so far in the context of learning an optimal way of interacting with users. This is what the following section is focused on. Other applications of RL in SDS for user simulations [Chandramohan et al., 2011] or to learn a reward function [Sugiyama et al., 2012] will be discussed in Part IV.

2.2.5 Frameworks for Spoken Dialogue Systems

The two main frameworks which have been used for RL-based dialogue management are MDP and Partially-Observable Markov Decision Processes (POMDP). MDP were the first to be used [Levin et al., 1997, Singh et al., 1999]. MDP offer a principled and simple framework for adaptive dialogue management. POMDP were suggested as a way to deal with the uncertainty coming from speech recognition and NLU into the agent's learning framework [Young, 2006, Williams and Young, 2007]. Indeed, in spite of the substantial progress on ASR, real-world operating SDS still suffer from a WER ranging between 15 and 30 percent [Black et al., 2010]. POMDP are a principled way of integrating this uncertainty. Then, another framework is the Module-Variable Decision Process (MVDP) framework [Laroche, 2010]. As said in the introduction, this framework was proposed to meet industrial criteria for production. Finally, Hierarchical RL (HRL) [Cuayáhuitl et al., 2007] on Semi-Markov Decision Processes (SMDP [Barto and Mahadevan, 2003]) was proposed to deal with large state spaces. The algorithms proposed in this thesis will be applied to MDP and MVDP dialogue managers so only these two frameworks are presented. However, these algorithms do not depend on the framework and can also be used with POMDP and SMDP.

The Module-Variable Decision Process (MVDP) framework [Laroche et al., 2009, Laroche, 2010, Laroche et al., 2010c] was more specifically formulated for hybrid handcrafted and statistically optimised SDS. As said before, an essential requirement when it comes to releasing a commercial SDS is VUI-completeness. The MVDP framework models dialogue management as a finite state machine. Modelling dialogue management with an automaton enables to encode prior knowledge on the temporality of the dialogue (e.g., the system should only greet the user at the beginning of the dialogue). Formally, an MVDP is defined by a tuple (M, V_M, A_M, T) where:
- M is the module space. A module is a node in the automaton.
- V_M is the space of *local contexts*. A local context is, for a module $m \in M$, the set $V_m \in V_M$ of variables which are relevant to this module.

- A_M is the set of possible actions (each module m has a set of possible actions $A_m \subset A_M$).
- $T \subset \mathbb{R}$ is the time measure, used in order to synchronise a module's decision with the distribution of a reward. In all that follows, time in MVDP will be measured as the number of *dialogue turns*.
The reward function is not included in the MVDP formalism. This is explained by the fact that rewards are event-based and not based on state-action or state-action-state tuples [Laroche, 2010]. For instance, the system might receive a negative reward for an ASR rejection or a user time out. However, in practice, it is possible to define a reward function on transitions (s, s') between two states s and s'. In effect, events are in practice encoded in states. It is the case in this thesis and when an algorithm is applied to the MVDP framework, the reward function is expressed on $S \times S$.

MVDP factorises learning into modules. Each module has its own state and action spaces. This framework was motivated by the fact that during a dialogue turn, several modules can be encountered, each making a decision according to a particular set of variables. For instance, a module can have to choose a question according to the system's belief of user expertise (novice, medium, or expert) and another one can have to make a decision depending on the supposed gender of the user. The contexts needed by the modules to make a decision are independent. Within the MVDP framework, one module optimises its policy according to the level of expertise and the other one, according to the gender. Prior knowledge can be easily encoded in the modules' local contexts and the complexity of learning is thereby reduced [Laroche, 2010].

This chapter focused on SDS design with an emphasis on dialogue management. It was shown that a dialogue manager should manage ASR misunderstandings, turn-taking timing, leave room for several degrees of initiative, and maximise cooperativity. Then, the main frameworks for RL-based management were presented. The following chapter presents and discusses different evaluation frameworks.

Chapter 3

Evaluating a Spoken Dialogue System

3.1 Overview

As for any other type of human-computer interface, the evaluation of SDS is multifold. Three types of evaluation frameworks can be found: the technical evaluation of the different components (from ASR to speech synthesis), the evaluation of usability, and the evaluation by users of the system and its components [Dybkjaer et al., 2004]. There is a global consensus on the important criteria to independently evaluate ASR (e.g., WER, concept accuracy), NLU (e.g.,. real time performance) and speech synthesis (e.g., natural aspect of the system's voice). As for the evaluation at the system level, the following evaluation criteria are often used: task success, number of problems encountered during the interaction, time to complete the task, and efficiency with which the user provided information to the system [Glass et al., 2000]. There is no consensus, on the other hand, concerning DM evaluation. Actually, a usability evaluation is often preferred to a specific DM evaluation because DM is closely tied to usability [Dybkjaer et al., 2004]. Möller [2000] defines usability as the "suitability of a system or service to fulfil the user's requirements. It includes effectiveness and efficiency of the system and results in user satisfaction", where user satisfaction reflects the user's perception of the dialogue features [Möller, 2005].

Therefore, measuring SDS usability has been an active research topic and many SDS usability evaluation frameworks have been proposed, for both academic [Walker et al., 1997b, Hone and Graham, 2000, Schmitt et al., 2012] and industrial systems [Evanini et al., 2008, Witt, 2011]. A long-lasting challenge has been to reconcile the two sectors by defining a common evaluation structure. If such structure existed, research results could be more easily integrated into industrial deployment processes [Paek, 2007]. Next section will cover the usability evaluation frameworks that have been proposed in the literature. These frameworks can be clustered into 4 types:

- Using a questionnaire analysed with statistical tools [Walker et al., 1997b, Hone and Graham, 2000, Hartikainen et al., 2004]
- Setting Wizard-of-Oz experiments to compare the performance of the system to human skills [Paek, 2001, Roque et al., 2006]
- Using crowdsourcing [Yang et al., 2010, Li et al., 2010, Jurčíček et al., 2011]

- Having expert annotators rate the dialogues [Evanini et al., 2008, Schmitt et al., 2012, Laroche et al., 2009]

These 4 types will now be described and discussed.

3.2 User Satisfaction Measurement

3.2.1 PARADISE

The PARAdigm for DIalogue System Evaluation (PARADISE) [Walker et al., 1997b, 1998, 1999, Walker, 2000] is a decision-theoretic framework that models usability in the way shown in Figure 3.1. The PARADISE

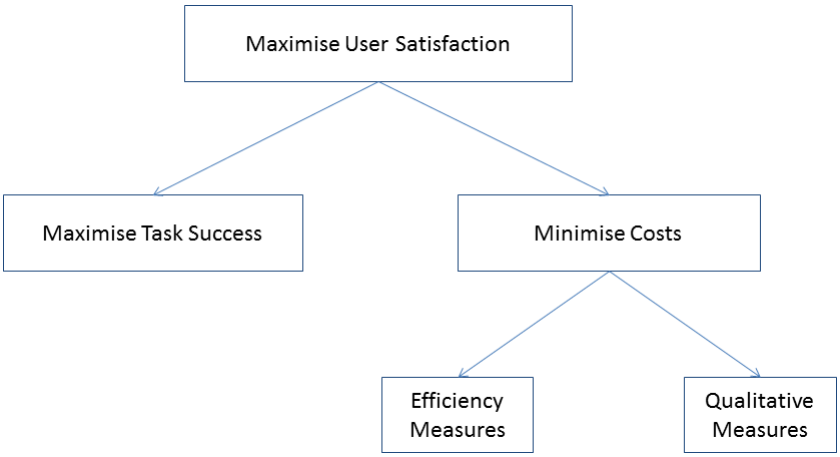


Figure 3.1: The decision theoretic model of usability in PARADISE.

framework was introduced in order to enable the comparison between cross-domain systems. In this framework, the usability of an SDS is measured as a trade-off between task success maximisation and dialogue costs minimisation. There are two types of dialogue costs: efficiency measures and qualitative measures. Efficiency measures are for instance the number of dialogue turns the system uses to complete the user's task. Qualitative measures comprise the dialogue features that play a role in the user's perception of the system such as the number of ASR rejections.

Task success is computed with the κ statistic [Cohen, 1960]. This statistic computes the probability of agreement $P(a)$ between two vectors y and $\hat{y}$ taking off the agreement that might occur by chance (probability $P(ca)$):

$$\kappa(y, \hat{y}) = \frac{P(a) - P(ca)}{1 - P(ca)}. \tag{3.1}$$

The probabilities are computed from a confusion matrix where the values collected by the SDS are compared to the values intended by the user. This computation of task success has the advantage of not being dependent on the domain. However, it only works for slot-filling tasks. The dialogue costs are not either depending on the domain. Therefore, PARADISE enables to compare several systems, designed for different slot-filling tasks.

The PARADISE framework goes even further in that it derives a user satisfaction metric from the decision schema in Figure 3.1. User satisfaction is modelled as a linear combination of task success and dialogue costs. The weights of the variables in the linear combination are learnt by performing least-squares multivariate linear regression. The PARADISE framework thus both provides a usability measurement framework and an estimator of usability. Once the linear regression has been performed, the system's usability can be estimated online with the linear estimator, without asking users to rate more dialogues. The dialogue costs can be automatically computed from the dialogue logs. Task success can be automatically computed or hand-labelled.

For an e-mail agent, Walker et al. [1999] proposed to measure user satisfaction with the following survey. The feature targeted by each question is between parentheses.

1. Was the system easy to understand in this conversation? (TTS Performance)
2. In this conversation, did the system understand what you said? (ASR Performance)
3. In this conversation, was it easy to find the message you wanted? (Task Ease)
4. Was the pace of the interaction with the system appropriate in this conversation? (Interaction Pace)
5. In this conversation, did you know what you could say at each point of the dialogue? (User Expertise)
6. How often was the system sluggish and slow to reply to you in this conversation? (System Response)
7. Did the system work the way you expected him to in this conversation? (Expected Behaviour)
8. In this conversation, how did the system's voice interface compare to the touch-tone interface to voice mail? (Comparable Interface)
9. From your current experience with using the system to get your email, do you think you'd use the system regularly to access your mail when you are away from your desk? (Future Use)

The survey was multiple-choice and the answers were mapped into a range of 1 to 5. A user satisfaction score was computed by summing the answers to all of these questions. Then, multivariate linear regression was performed and returned the following estimator of User Satisfaction (US):

$$\hat{\text{US}} = 0.27\ \text{task\_completion} + 0.54\ \text{mean\_recognition\_score} - 0.09\ \text{user\_barge\_in\_rate} - 0.15\ \text{rejection\_rate}$$

Task completion and the mean ASR score have been shown to play a prominent role in user satisfaction for many SDS [Larsen, 2003, Walker et al., 2002].

Some critics have been formulated towards the PARADISE framework. First, the linear parametrisation of user satisfaction suffers from a lack of theoretical and experimental grounding [Larsen, 2003]. Besides, the computation of the user satisfaction score done by summing the scores given to the target features in the previous questionnaire also lacks theoretical grounding [Möller, 2005]. In order to bypass this difficulty, following [Hone and Graham, 2001], Hajdinjak and Mihelic [2007] used a questionnaire with items which were directly linked to the SDS component being evaluated. For instance, if dialogue management is evaluated, answers to some questions such as questions 1 and 2 should not be accounted for in the evaluation. Another hindrance to the usage of PARADISE is the κ statistic. Its computation can become burdensome when some attributes can take many different values. In addition, this statistic is not adapted to cases where the attributes whose values are to be collected during a dialogue are not strictly defined in advance. These drawbacks imply that the κ

statistic is often replaced by a subjective evaluation of task completion, which also happens to be a better predictor of user satisfaction [Larsen, 2003]. However, subjective evaluation of task success defeats the purpose of building an automatic estimator of user satisfaction. Finally, the PARADISE questionnaire has not been subject to a psychometric evaluation. This last point has led to the design of SASSI (Subjective Assessment of Speech System Interfaces) [Hone and Graham, 2001] which is a usability evaluation questionnaire that has been validated.

3.2.2 SASSI

SASSI [Hone and Graham, 2001] was built by adapting usability evaluation questionnaires for non-speech interfaces to vocal interfaces. However, it did not comprise speech synthesis. The SASSI questionnaire was applied to 4 systems dedicated to the following tasks: phone dialling, bank account managing, in-car radio/climate controlling and stereo handling. A principal component analysis was applied to the answers, which highlighted six factors: system response accuracy (the system was precise, acted as desired by the user), likeability (the SDS was pleasant to use, will be used again in the future), cognitive demand, annoyance (the system was monotone, irritated the user), habitability (the user knew what to say), and speed (response delay, dialogue length).

Möller et al. [2007] modified the SASSI questionnaire in order to integrate speech synthesis. The principal component analysis highlighted six factors, which partly overlapped with the ones put forward by the SASSI analysis: acceptability (future use, overall quality), transparency (the user knew what to say, felt in control of the interaction), interaction efficiency (speed, dialogue length), cognitive demand, system cooperativity, and task success.

As noticed by Paek [2007], the SASSI questionnaire indicates the items that should be measured in a usability evaluation but it is unclear how these measures could be used to improve the system.

3.2.3 SERVQUAL

Hartikainen et al. [2004] proposed to use the SERVQUAL (SERVice QUALity) method to proceed to the evaluation of SDS usability. SERVQUAL was developed in the marketing domain to evaluate the quality of a service provided to the user. It consists of measuring the difference between the users' expectations and their perceptions of the service. Hartikainen et al. adapted the original questionnaire to spoken dialogue. For instance, the evaluation of the staff's appearance was replaced by the evaluation of the system's voice.

The utilisation of SERVQUAL for spoken dialogue was motivated by the fact that most SDS, including the commercial ones, provide a service to the user. A drawback of SERVQUAL is that the user's perception of the service can be very sensitive to manipulation (depending on the questionnaire) and the advisement of this perception can lead to an erroneous evaluation of the system's capabilities [Paek, 2007].

3.2.4 Comparison to human gold standard

A Wizard-Of-Oz (WOZ) experiment with an SDS consists of asking an operator to play the role of dialogue management. The operator receives the user dialogue act returned by NLU and selects the next action of the system which is then passed on to NLG. It has been proposed to measure some aspects of system usability by measuring the proximity of the system's behaviour with the one of a human who would act under the same level of uncertainty [Paek, 2001, Roque et al., 2006]. For instance, Paek [2001] compared the task completion rate of

an SDS and a human for several WER of speech recognition. This metric allows to compare disparate systems since the task domain is not involved in the evaluation. WOZ experiments could be used to measure system usability by asking users to rate both SDS and human performances. However, this means that the experiment should be coupled with an evaluation questionnaire. The method does not provide *per se* a principled way to measure usability. In addition, WOZ experiments are quite expensive to set and it is not always sure that a human can be considered as a gold standard. Machine learning makes more sense than human emulation for some dialogue management choices such as the first strategy to use at the beginning of a dialogue. It is likely that a human will try different strategies and then choose one based on a very limited experience. Machine learning, on the other hand, enables to explore and draw accurate conclusions from a wider range of experiences.

3.2.5 Overall Rating

In the evaluation frameworks presented so far, users were asked to fill in a questionnaire. Each item of the questionnaire was dedicated to a specific aspect of the interaction. In addition, an overall evaluation of the system was included. For instance, the modified SASSI questionnaire proposed in [Möller et al., 2007] had the following items: "overall impression of the interaction with the system" and "overall, I am satisfied with the system", evaluated on a 5-point Likert scale (bad, poor, fair, good, excellent). These two items were strongly related to the acceptability dimension. Measuring the overall satisfaction[1] allows to easily compare systems [Laroche et al., 2011] or even different dialogue management strategies with the same system. It is also an easy way to get a general idea of the system's acceptance by the users. The overall rating can be used as a single value to be reported as a measure of the *system's performance*.

It is important to pursue system evaluation once the system is online. Indeed, the system's modules are likely to be confronted to user behaviour and utterances which have not been previously observed. Once the system has been released, it is important to have a way to inform the designer if the overall evaluation reaches a given low limit (e.g., 7 if the usability is evaluated on a scale of 1 to 10). The designer can then look more carefully at the dialogues with an insufficient overall usability score and identify the issues encountered by the system during these dialogues. The online tracking of a system's usability requires either that an automatic estimator is learnt from a corpus of user-evaluated dialogues or that each user of the system is asked to evaluate the system after interacting with it. In both cases, a single value can be computed. Indeed, users cannot be reasonably asked to fill in a questionnaire after each interaction. Anyway, the first option is most often pursued. One reason for it is that online user evaluation suffers from strong biases. Indeed, users who are not satisfied with the system are more likely to participate in the evaluation. The performance of ASR, NLU, NLG, and speech synthesis are already known and can be measured with generic metrics such as WER, concept accuracy and naturalness of speech synthesis. Dialogue management evaluation, on the other hand, must be pursued online, to measure the system's capability of adapting to different user profiles. The profiles of the end users are likely to be different than the ones of the users specifically recruited to evaluate a system. Therefore, if the behaviour of the dialogue manager is based on the evaluated corpus, it is likely to be maladjusted and to need to be refined according to the dialogues with the end users.

As a consequence, online evaluation of SDS should be targeted at dialogue management. The usability questionnaires presented so far did not specifically target dialogue management. Actually, overall user satisfaction ratings are biased by the user's environment or the perception of aspects unrelated to dialogue

[1] whether it is with an explicit item, or by summing answers to different items as in PARADISE

management such as the voice of speech synthesis [Möller, 2000, Schmitt et al., 2011]. A way of overcoming these biases is to appeal to experts instead of users. The experts listen to the dialogues and fill in a questionnaire or give an overall rating to the system according to the system's management of the interaction [Evanini et al., 2008, Schmitt et al., 2011]. This requires nevertheless that the experts are trained in order to avoid a great variability in their rating styles. In all that follows, following Schmitt et al. [2011], user ratings will be designated as User Satisfaction (US) and expert ratings as Interaction Quality (IQ).

3.3 Interaction Quality Measurement

3.3.1 Caller Experience

The Caller Experience (CE) metric was designed for commercial SDS [Evanini et al., 2008]. Experts were asked to give a score on a scale of 1 to 5 to each dialogue according to the quality of the interaction with the user. CE accounts for the quality of speech recognition, the ability of the system to identify the user's goal, and the appropriateness of the system's answers. The experts received the following guideline for the evaluation: compared to a human-human interaction on the same subject and with the same amount of information, did the system act as well as possible? Evanini et al. showed that this metric was reliable as different trained experts had a very high agreement rate (κ of around 0.80, see Equation 3.1). However, it is unclear what the CE metric really measures. The criteria chosen to evaluate the CE have not been proven to be closely related to system usability or any other known metric.

Crowdsourcing

Like WOZ experiments, having dialogues rated by experts is costly and time-consuming since it requires training the experts. Yang et al. [2010] used the crowdsourcing platform Amazon Mechanical Turk (MTurk, www.mturk.com) in order to have a large number of dialogues evaluated for a reasonable price. This platform allows to post a set of HIT (Human Intelligence Tasks). Workers, who on this platform are commonly called *turkers*, can then apply to complete the tasks in exchange of a monetary payment. Yang et al. showed that the turkers' work could be verified. For their experiment, turkers were asked to classify each dialogue into one of the following three categories: task complete, task incomplete, request that cannot be managed by the system. The classification was then compared to an automatic classification that had been performed beforehand. Another way to verify the workers' answers was by checking the coherence in their answers. For instance, one item the workers had to assess was whether the system succeeded in providing the information sought by the user. A worker who answered *yes* to this question should not have classified the dialogue in the category of requests which could not be managed by the system. The agreement rate between evaluators was also a good indicator. Yang et al. observed an agreement rate of 60% for some items. The items which had the lowest agreement rate were the subjective criteria which were related to the perception of system quality by the turker. Li et al. [2010] kept from this study the evaluation criteria with the highest agreement rate: did the system provide to the user the information requested?

Another possibility is to ask turkers to interact with the system and then fill in an evaluation questionnaire. Jurčíček et al. [2011] gauged the strengths and weaknesses of such process. They argue that the evaluation of the system would be more reliable if the SDS provided a service that the turkers really needed. The workers would not have to put themselves in a simulation situation and follow a predefined scenario. In addition, in a scenario-based situation, the evaluators tend to be more indulgent with the system. This point, coupled

with the relatively low agreement rate of 60% for turkers rating dialogues, and the need to come up with a verification framework make the use of MTurk quite infrequent at the moment. The development of this evaluation technique would require a rigorous generic protocol be set.

3.3.2 Dialogue Turn-Level Rating

Two types of dialogue turn-level evaluations can be found. Witt [2011] proposed to adapt the CE metric to evaluate the quality of an interaction on a turn level basis. To do this, the author asked a trained expert to rate calls on a scale of 1 to 5. From these ratings, a dialogue turn-level estimator of CE was learnt. The second type was adopted by Schmitt et al. [2012] who directly asked the experts to give a score on a scale of 1 to 5 after each system-user exchange. Turn-level evaluations allow to dynamically adapt the system's behaviour. For instance, based on a critical threshold for the estimation of CE, the call could be transferred to an operator or switched to DTMF [Witt, 2011].

Having an automatic estimator of US or IQ not only enables to dynamically adapt the system's behaviour, it also saves the cost of recruiting and training experts to evaluate the dialogues. The next section reviews the models that have been proposed so far for the automatic estimation of the usability of an SDS.

3.4 Automatic Estimation of user satisfaction and interaction quality

Evanini et al. [2008] learnt a decision tree from the dialogues whose CE had been evaluated by experts. The class of CE of a dialogue was decided according to the following features: system ability to identify the user's goal, number of ASR errors, number of times the user asked to be transferred to an operator, outcome of the dialogue (task complete vs. call transferred to an operator). The agreement rate between the decision tree and the experts' evaluations ranged between 75-80%. It is difficult to evaluate online some of the criteria used to build the decision tree. For instance, task completion cannot always be automatically assessed online.

To compute online a turn-level estimation of CE, Witt [2011] tracked the events that occurred during the dialogue. These events were ASR rejection, user time out, user intention misunderstood, user contradicting the system, and user asking to be transferred. At time t, the CE metric was computed according to its value at time $t-1$ and the weight of the latest event, as shown in the following equation:

$$\text{CE}_t = d \times \text{CE}_{t-1} + \text{ weight of the latest event.} \tag{3.2}$$

Like in RL, the parameter d is a discount factor, used to reduce the importance of the furthest events in time. The weight of each event was learnt on a set of annotated dialogues with three different systems. The weights were learnt from a set of empirical rules such as: CE should be above the threshold after two user contradictions. These rules were turned into a set of equations from which the weights and the threshold were computed. The agreement rate with the experts was of 64%. By manually tuning the weights, the agreement rate reached 76%. In practice, the set of weights that was kept was the one that optimised the trade-off between CE value and call success. Several thresholds were tested with a call-routing SDS. A threshold of 4.9 enabled to optimise both CE and call automation. This estimator entirely depends on the rules defined by the designer. In addition, the set of rules has to be redefined for each new SDS implementation. Another drawback of this estimator is that hand-tuning the parameters seems to dramatically change the results both in terms of agreement rate and call success.

Many other models have been proposed to compute an automatic estimator of IQ or US. Engelbrecht et al. [2009] used Hidden Markov Models (HMM) to represent the evolution of US throughout a dialogue. Users were asked to evaluate the dialogue after each exchange with the system. The evolution of US was then represented as an HMM where the states were user judgements and the observations comprised understanding errors and confirmation types. The problem here is that understanding errors have to be manually annotated. Besides, the dialogue was paused after each exchange for the user to give a rating, which affects some crucial aspects of a human-computer interaction such as dialogue duration. Higashinaka et al. [2010] compared HMM and conditional random fields to estimate turn-level IQ. Hara et al. [2010] built an N-gram model to estimate US at a dialogue level. In these two works, the estimator was computed based on user and system dialogue acts. The models used become intractable with continuous variables such as dialogue duration. However, approximation methods can be used such as particle filters.

As said in the previous section, rather than asking users to rate a dialogue on a dialogue-turn level, Schmitt et al. [2011, 2012] asked experts to give a score on a scale of 1 to 5 for each exchange. Each dialogue started with a score of 5. Schmitt et al. [2011] trained a Support Vector Machine (SVM) on 200 evaluated dialogues. A drawback with SVM for classification in this type of application is that the order between the labels is not accounted for. For instance, the model does not take into consideration that for an actual rating of 1, predicting 2 is better than predicting 5.

3.5 Positioning

Two issues have been identified so far: choosing a dialogue management evaluation strategy and a model to automatically estimate user satisfaction or interaction quality. As discussed before, overall ratings are a link between the two problems.

For the first issue, it is chosen here to ask users to fill in a questionnaire targeted at dialogue management and to give an overall rating after each dialogue. Two choices are underlying this decision. The first choice is to evaluate US and not IQ. The subjectivity biases mentioned before can be restrained by orienting the questions in the evaluation questionnaire solely towards dialogue management and not including for instance the naturalness of speech synthesis [Hajdinjak and Mihelic, 2007]. Besides, the users evaluating the system are volunteers who are asked to engage in a scenario-based dialogue. These users do not have any personal stakes in performing the dialogue, which reduces the difference between user rating styles. Finally, the volunteers are Orange employees already familiar with dialogue systems. They can use their experience as a baseline to score the dialogue management strategies. The second choice is to rate the dialogue only at the end. As said before, it is not desirable to halt the dialogue after every exchange in order to ask the user to give a score. Besides, from an RL perspective, it is not necessary to have an estimation of US after each exchange. Indeed, RL is specifically convenient for learning tasks with delayed rewards.

For the second issue, several models will be tested and compared in Chapter 9. A crucial point is the set of variables used to perform the estimation. In accordance with the guidelines exposed in the introduction, only automatically computable variables will be kept. Some studies have strived to identify the dialogue features which enable to predict when a dialogue is problematic [Walker et al., 2002, Schmitt et al., 2008, 2012]. The features used in the experiments are inspired by these studies and will be fully listed in Chapter 5.

This chapter discussed SDS evaluation and stated the positioning adopted for the experiment carried during this project and suggested for the methodology proposed in the introduction. The following part will describe the tasks which will serve to test the different algorithms proposed in the thesis and then the appointment

scheduling systems.

Part II

Test Domains

This part presents the test domains for the algorithms which learn a state space representation and a reward function from rated and annotated dialogues. The first chapter describes the already existing RL tasks and spoken dialogue simulators. The second chapter proposes two new SDS for appointment scheduling. The appointment scheduling systems were designed for several reasons.

The SDS NASTIA is used to collect rated dialogues. It also serves as illustration for the scalability of the state space representation proposed in this thesis with as much as 120 features being extracted from its dialogue logs. The simulator is an interesting setting because it has to find a balance between user satisfaction and its own task completion. This setting is likely to be found in an industrial context where an SDS should provide a service for the user but also respect some business constraints. The already existing SDS used for this thesis provide other benchmark for the reward inferring algorithms on corpora of small sizes and the RL task known as mountain car is convenient for testing the state space representation on a continuous space with few dimensions, which makes it easy to interpret the learnt policy.

Chapter 4

Existing Dialogue Systems and Reinforcement Learning Tasks

In this chapter are presented the different test domains used for the algorithms proposed in this thesis. The state space representation is tested on a classical RL task known as *mountain car* and on the appointment scheduling simulator. The reward function inference algorithms are tested on three SDS: an appointment scheduling system named *CLASSiC System 3*, a light version of *TownInfo* which is an SDS helping users find restaurants, and the appointment scheduling simulator. Estimators of user satisfaction are tested on the *LEGO corpus*. The entire methodology in Figure 1.1 is illustrated on the appointment scheduling system NASTIA and on the simulator. These two systems are presented in the next chapter.

4.1 Dialogue Systems

4.1.1 CLASSiC System 3

Description

The CLASSiC EU FP7 project[1] studied robust statistical learning in SDS [Laroche et al., 2011]. During this project, several appointment scheduling systems were built (referred to as Systems 2, 3, and 4 in [Laroche et al., 2011]). These systems enabled users to schedule an appointment with a technician in case of landline dysfunction. Among them, System 3 is a French-speaking system cast as an MVDP. The purpose of learning in this SDS is to assess the influence of the prosody of speech synthesis on user satisfaction. For each SDA, the dialogue manager has to choose an intonation between three styles: calm, neutral, and dynamic.

The negotiation strategy of the system is hard-coded (see Figure 4.1). In Figure 4.1, a node of the graph stands for a module and an arrow from one module to another indicates a possible transition between these modules during a dialogue (possibly after 0 or several dialogue turns). The system starts each dialogue by proposing to the user its first availability (module 1). Then, if the user rejects the proposition, the system asks her to give her first availability (module 3). If this is not a free slot in the system's calendar, the system proposes its next availabilities (module 2) until an appointment is booked (module 7) or the system has no more propositions to make (module 8). When a user proposes a date, the system asks for a confirmation through

[1] Computational Learning in Adaptive Systems for Spoken Conversation, http://www.classic-project.org/

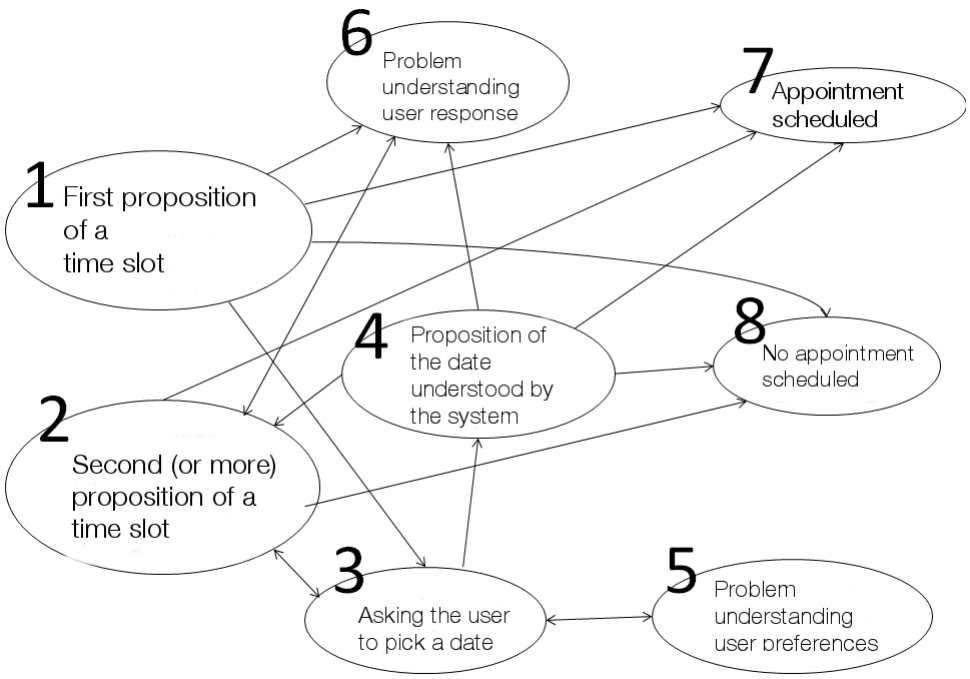


Figure 4.1: Dialogue logics of CLASSiC System 3.

module 4. Two error repairing modules (modules 5 and 6) notify the user that she has not been understood (in case of an ASR rejection) or heard (in case of a time out).

Each dialogue is a succession of dialogue phases. All the modules in Figure 4.1 are in separate phases. The classical course of a dialogue phase starts with a statement by the system, followed by an answer from the user. Transition to the next phase depends on user behaviour. Figure 4.2 illustrates the phase in System 3 which contains module 3. An example of system-user exchange is: the system asks the user to propose a date for an appointment, the user answers with a date and then the system transitions to another phase where the user will be asked for a confirmation of the date understood by the system. System 3 was designed with Disserto, Orange's software to build an SDS as an automaton. Formally, System 3 is modelled by the following MVDP:
- $M = \{m_1, ..., m_8\}$ is the set of eight modules.
- $\forall\, m_i \in M, V_{m_i} = \emptyset$.
- $\forall\, m_i \in M, A_{m_i} = \{$calm, neutral, dynamic$\}$.
- $\gamma = 0.95$.
The reward function was handcrafted and it gives a reward equal to 10 when module 7 is reached, -10 when module 8 is reached and -1 after each speech recognition rejection or user time out (modules 5 and 6). The set of local contexts is the empty set, which means that a module m_i does not account for any specific context to choose an action in A_{m_i}. The environmental dynamics (here user behaviour) were unknown and no model was built i.e., learning was model-free [Laroche, 2010].

CLASSiC System 3 was evaluated on 740 dialogues which were performed by volunteers who had to follow a scenario [Laroche et al., 2011]. 12 scenarios were uniformly distributed among the participants. Each

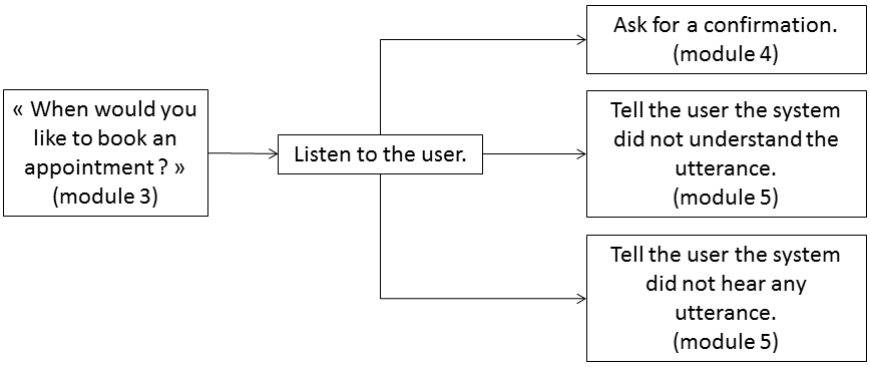


Figure 4.2: Example of a dialogue phase in CLASSiC System 3.

scenario was associated with a two-week calendar. In the example given in Figure 4.3, user availabilities are in green. Each scenario was characterised by a system difficulty and a user difficulty (on a scale of 1 to 4). These levels of difficulty were computed according to the first common availability between system and user calendars. For example, if the first availability of the user was the second availability of the system, then the scenario was of difficulty 1 for the user (if the user gives her first availability, it will be accepted by the system) and 2 for the system (the system will have to propose its first two slots to set an appointment). In each scenario, only one time slot was vacant for both the user and the system.

After each dialogue with System 3, the user was asked to fill in the following questionnaire:

1. Did you book an appointment?
2. Was the appointment booked during one of your available slots?
3. When did you book the appointment?
4. During your interaction with the system, the system understood what you said.
5. During your interaction with the system, you understood what the system said.
6. During your interaction with the system, the system's voice was pleasant.
7. During your interaction with the system, was it easy to set an appointment?
8. Explain the difficulties that were encountered.
9. This interaction has prompted me to use such systems in the future.
10. What grade would you give to this interaction on a scale of 1 to 10, 10 being the highest grade?
11. Do you have any other remarks or comments?

Questions 4 to 7, as well as question 9 were evaluated on a six-point Likert scale (completely disagree, disagree, mostly disagree, mostly agree, agree, completely agree).

Figure 4.3: Example of user calendar in the CLASSiC experiment. The user's available time slots are the green ones.

Usage

The corpus of evaluated dialogues collected with System 3 is used to illustrate the two algorithms inferring a reward function. For this purpose, the overall evaluations (question 10 in the questionnaire) are used as performance scores. The corpus provides extended logs for each dialogue from which can be extracted different features of the dialogue such as the sequence of module-action pairs visited, the rewards obtained, the user and system dialogue acts, the number of dialogue turns,... The challenges with this corpus are the fact that it entails dealing with US scores and a corpus of a relatively small size (740 dialogues). Nevertheless, the size of the module-action space (27) makes it convenient for RL.

4.1.2 TownInfo

Description

The DIPPER architecture [Bos et al., 2003] is based on the Open Agent Architecture (OAA, [Martin et al., 1999]) which allows to integrate, in a distributed environment, agents possibly written in different languages and running on different platforms. Agents synchronise *via* the Inter-agent Communication Language (ICL). An SDS built with this architecture contains an agent for speech recognition, one for DM, one for speech synthesis, and other agents for semantic interpretation and generation. The DM provided with DIPPER implements the information state approach[2].

The TownInfo system provides information about a city's restaurants, hotels, and bars [Lemon et al., 2006]. Its DM is based on the one provided with the DIPPER architecture. The SDS that will be used in this thesis is a light version of TownInfo, which will be referred to as *LTI* (Light TownInfo). LTI only provides information

[2]The DIPPER software is freely available for research purposes at http://www.ltg.ed.ac.uk/dipper/.

concerning restaurants. DM is modelled as a slot-filling task: the user has to specify a price range, an area, and a type of food for the system to return a result. The possible values for each slot are given in Table 4.1. In the

Type of food	Price range	Area
Italian	Moderate	Central
Indian	Expensive	North
Chinese	Cheap	South
		West
		East

Table 4.1: Possible values for the slots of LTI.

test performed with LTI, RL is MDP-based. The MDP is defined as follows:
- $S = \{$price_range_{empty, known, confirmed}, area_{empty, known, confirmed},
type_of_food_{empty, known, confirmed}$\}$.
- $A = \{$greet, AskASlot, ExplicitConfirm, ImplicitConfirmAskASlot, closeDialogue$\}$.
- P is unknown.
- R gives an immediate reward equal to 0 after each dialogue turn and a final reward equal to -3#EmptySlots + 0.25#CorrectSlots - 0.75#WrongSlots - 0.015#Turns.
- $\gamma = 0.99$.

The state space is an example of summary space. Instead of keeping in the state space the value for each slot (e.g., type_of_food=Italian, price_range=moderate, area=central), the system only takes into consideration the state of its knowledge concerning the slot, namely, whether it is unknown, known or confirmed. An example of summary state is: type_of_food=known, area=confirmed, price_range=empty. The action space is also defined on a summary space. Actions are not differentiated according to the slot involved. For instance, AskSlot_price_range, AskSlot_area and AskSlot_type_of_food are all summarised into AskASlot. Hand-crafted heuristics were used to map summary actions back into the master space (for instance, AskASlot can only be applied to empty slots). The reward function was here again handcrafted. The system's final reward strongly decreases with the number of empty slots in order to encourage the system to ask for all the slots. The number of turns is to be minimised. All in all, this function encourages the system to maximise task success while minimising dialogue duration.

Usage

LTI is used to compare the learning speed of the dialogue manager with the different reward functions that our algorithms infer from data. The comparison is made on 600 simulated dialogues. User behaviour is simulated according to the Bayesian method proposed by [Pietquin et al., 2009]. This method models user behaviour as a Bayesian network to simulate dialogues at the intention level, including grounding behaviours. Each simulated user has a goal, e.g., find a cheap Chinese restaurant in the southern part of town. The user can then inform the system about the value of a slot, confirm or negate if the system asks for a confirmation (implicitly or explicitly). The parameters of the Bayesian network were trained on the 1234 human-machine dialogues which are described in [Williams and Young, 2007]. The trained Bayesian network-based simulator was shown to perform better than one whose parameters had been set by an expert based on simple dialogue statistics such as the average number of turns. To make the dialogues even more realistic, speech recognition errors were also simulated [Pietquin and Renals, 2002].

To compare inferred reward functions, one has to first learn the functions on a corpus of evaluated dialogues and then learn policies with these functions on another evaluated corpus. Having a user simulator is convenient to produce as many dialogues as necessary for convergence. The purpose of the test on LTI is to compare learning speed with the reward function of LTI described in the previous section and the inferred ones. For this purpose, since the object of the experiment is not performance score prediction, real evaluation scores are not needed. Besides, the summary RL model of LTI enables to learn a policy with a relatively low number of simulated dialogues.

4.1.3 LEGO corpus

Description

CMU's Let's Go system provides local information on bus schedules in the city of Pittsburgh [Raux et al., 2003]. The LEGO corpus is a set of 200 dialogues with Let's Go which were annotated and evaluated by three experts [Schmitt et al., 2012][3]. The experts were asked to give an IQ score on a scale of 1 to 5 after each system-user exchange. Each dialogue starts with an IQ of 5 and then the expert can degrade this score according to the quality of the system's DM. In total, 5282 system-user exchanges were evaluated. Schmitt et al. [2011] showed that the median of the three scores had a better correlation with the ratings than the mean. The median is thereby used as IQ score for each system-user exchange.

Each system-user exchange in the corpus is described as a set of features, which are exhaustively listed in [Schmitt et al., 2012]. Automatically computable and manually annotated features (e.g., the emotional state of the user) are included. The feature set contains parameters related to ASR (confidence score, rejection,...), NLU (user dialogue act, semantic parsing,...) and DM (system dialogue act). Moreover, the values of the features are computed on three different levels: the value for the current dialogue turn, the mean value up to the current dialogue turn and the mean value over the last three exchanges.

Following the third guideline presented in the introduction, the inferred reward function and space representation should be usable on-line. As a consequence, only the automatically computable features in the corpus are kept. These features are described in Table A.1 in Appendix A.

Usage

The LEGO corpus serves to compare several IQ estimators. 10-fold cross validation is applied: the estimators are learnt on 90% of the corpus and tested on several metrics on the remaining 10% dialogues. The most efficient estimator is integrated in one of the reward function learning algorithm. Two features in Table A.1 are not numerical: User Dialogue Act (UDA) and System Dialogue Act (SDA). UDA can take values among UDA_CONFIRM_DEPARTURE, UDA_LINE_INFORMATION, and so on. For these cases, noting n_a the number of possible labels, the feature was split into $n_a - 1$ variables, each variable being a boolean. SDA was split into 27 variables and UDA, 21. In total, 67 features were kept. Feature values were centered and normalised.

An advantage brought by the usage of this corpus is that the features in Table A.1 are quite generic so the performance of the estimators on this corpus gives a good idea of their performance on other domains.

[3]The LEGO corpus is freely available for research purposes at http://www.uni-ulm.de/in/nt/staff/research-assistants/ultes/nt-ds-lego.html.

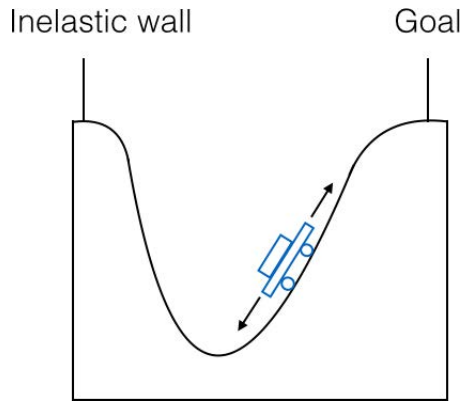


Figure 4.4: The mountain car configuration.

4.2 Other Reinforcement Learning Tasks

4.2.1 Mountain Car

Description

The *mountain car* task consists of getting a car on top of a hill. At first, the car starts in a valley with not enough thrust to just drive up the hill. In order to get to the top, the car first has to drive up the opposite hill and, once enough thrust has been gained, drive up the hill whose top must be reached [Moore and Atkeson, 1995]. A schematic display of this configuration is given in Figure 4.4. An additional difficulty is added: the car has to park in some region at the top of the hill. This task, named *parking car*, is more difficult than the classical mountain car task because the car has to stop so it should not arrive too fast in the goal area. Parking car is modelled as a continuous episodic MDP:
- $S = \{\text{position, velocity}\}$.
- $A = \{-1, 0, 1\}$.
- P is unknown.
- R gives an immediate reward equal to -1 for each action taken by the car except when the goal position and velocity are reached. In this case, the car receives a reward equal to 0.
- $\gamma = 0.95$.

The car's state is described by its position x and velocity $\dot{x}$. The position is comprised between -1 and 1 and the velocity between -2 and 2. When a limit position is reached, the car is reset to this limit position with a null velocity, simulating an inelastic collision. An episode ends when the car has reached a position between 0.5 and 0.7 with a velocity $|\dot{x}| \leq 0.1$ or after 1000 iterations. The car must choose an action between applying a horizontal force equal to -4N (action -1), 0N (action 0) or 4N (action 1). Another configuration with less powerful actions (-1N, 0N, 1N) will also be explored. Given an action a_t, the velocity and position of the car are updated according to the following equations:

$$\begin{aligned} \dot{x}_{t+1} &\leftarrow \dot{x}_t + 0.001a_t - 0.0025cos(3x) \\ x &\leftarrow x + \dot{x} \end{aligned} \tag{4.1}$$

Usage

The mountain car task is a good illustration of RL's aptitude of dealing with delayed rewards. Indeed, the car first has to drive up the opposite hill and get negative rewards before it has enough power to gain the opposite hill and get a null reward. The first negative rewards are necessary to achieve the task. This domain is also a good benchmark for exploration vs. exploitation strategies. In effect, the car has to explore the ranges of positions and velocities in order to find a sequence of actions that will get it to the top of the hill.

The mountain car domain is used to test learning with the state space representation proposed in Chapter 7. It is a standard benchmark for continuous RL [Ratitch and Precup, 2004, Rasmussen and Kuss, 2004, Timmer and Riedmiller, 2007]. Another advantage is that the state space having only two continuous dimensions, the state space model and its associated policy are easy to interpret.

This chapter have presented different RL-based SDS and an RL benchmark which are used to test different algorithms. New SDS were also designed to further explore the methodology. They are described in the following chapter. First, the design of NASTIA and the user experiment carried out with this system are explained. Then, the appointment scheduling simulator is described.

Chapter 5

The Appointment Scheduling Systems

This chapter describes NASTIA [El Asri et al., 2014d], the Negotiating Appointement Setting Interface that was designed during this thesis, as well as DINASTI (Dialogues with a Negotiating Appointement Setting Interface) [El Asri et al., 2014b], the corpus of dialogues collected with NASTIA, and the appointment scheduling simulator. First, previous appointment scheduling systems are discussed.

5.1 Previous Work on Spoken Dialogue Systems for Appointment Scheduling

5.1.1 The SCHEDULER

The SCHEDULER [Lacson, 2004] is an SDS dedicated to the management of medical appointments. The first action performed by the SCHEDULER is to ask for the patient's name to check that she is already in the records of the medical facility. Then the user chooses to create, cancel, or check an appointment. To create an appointment, the user has to indicate the practitioner's name and a day. If some of these criteria are missing and the system cannot query the database, the user is asked for the missing items. Users may also specify a time preference and if they do not, the system simply proposes its first availability for the given day. The SCHEDULER was evaluated according to the three following parameters: task success, task ease, and difficulties encountered during the dialogue. Nevertheless, the evaluation only concerned 15 calls which were not scenario-based so it is not possible to draw any reliable conclusion on the usability of this system.

5.1.2 Systems Designed During the CLASSiC Project

The CLASSiC project gave birth to three systems enabling users to schedule an appointment with an engineer in the case of a dysfunction of their landlines. These three systems are referred to as Systems 2, 3, and 4 in [Laroche et al., 2011]. System 3 was described in the previous chapter.

System 2 [Jurčíček et al., 2010] was a state-of-the-art POMDP-based SDS [Sondik, 1971, Williams and Young, 2007]. After each user input, the dialogue manager received the n-best list of semantic hypotheses, updated its belief state, and chose its next dialogue act accordingly. With this system, the user could provide one or several constraints such as day of week, day of month, etc. Then the system asked the user to refine her constraints until it could identify a unique available slot or it determined that there was no available slot matching these constraints. The system could also provide information about its available slots given the user's constraints or offer an alternative if the constraints did not correspond to any available appointment.

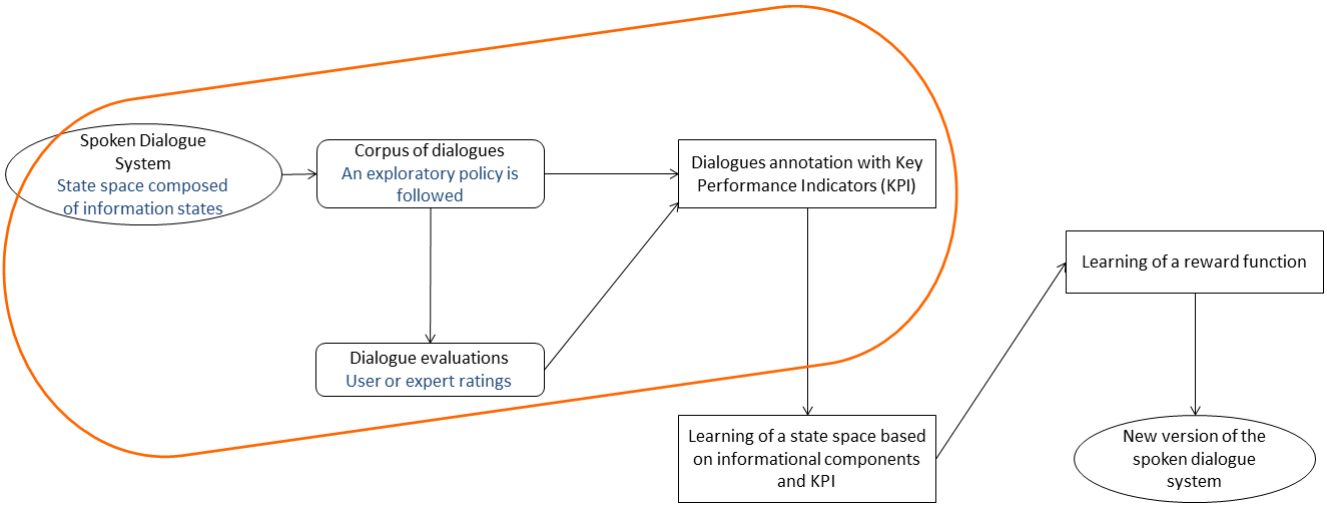


Figure 5.1: First part of the design of an RL-based SDS whose reward function and state space are to be learnt from data.

Contrary to systems 2 and 3, System 4 was not RL-based. The system either proposed a time slot or asked for different constraints such as week, day, half-day until it was able to make a proposition matching the constraints or reject the constraints. The system chose between the two strategies based on the number of remaining slots. If there were 2 or fewer time slots, it proposed a time slot. Otherwise, it asked the user for her constraints. If so, the system asked the user to specify turn by turn a day, a week, and a half-day. First, the system asked for the most restricting parameter, i.e., the one that minimised the number of questions to ask to the user given the current system calendar.

These three systems were tested and compared on scenario-based dialogues following the methodology explained in the previous chapter. These experiments enabled us to point out the parts of the appointment scheduling process that needed to be improved. The design of NASTIA resulted from the analysis provided in [Laroche et al., 2011].

5.2 NASTIA

Figure 5.1 reproduces the SDS design methodology presented in the introduction. This chapter illustrates the first part (circled in red) of the design: from defining the SDS with informational states to collecting dialogues, having them evaluated, and annotating them with KPI.

NASTIA is a French-speaking SDS which studies the appointment scheduling task, a line of research started by Lacson [2004] and continued during the CLASSiC project [Laroche et al., 2011]. NASTIA's task is to schedule an appointment with a user who needs the intervention of an engineer on site. Appointment scheduling is both a slot-filling and a negotiation task. The SDS can choose between three negotiation strategies, more or less conservative, depending on the course of the dialogue. Other choices such as confirmation strategies are also made with RL.

5.2.1 Automatic Speech Recognition and Natural Language Understanding

Automatic speech recognition was performed by Nuance's recogniser[1]. The range of expressions recognised by ASR was restricted to a set of manually defined tags which are relevant for appointment scheduling (expressions to say a date, a time period, agree or disagree). After each user utterance, the recogniser returned the most likely hypothesis with the corresponding confidence score between 0 and 100, e.g <*"Tuesday", score=78*>. The NLU module then interpreted this hypothesis according to grammar rules which were handcrafted and returned a dialogue act. For instance, the previous hypothesis would become <*day=Tuesday, score=78*>. The user dialogue acts recognised by NLU are listed in Table C.1 in Appendix C.

5.2.2 Natural Language Generation and Speech Synthesis

NLG was template-based. Some utterances were static, e.g., the welcoming prompt. Others had a dynamic part, e.g., the ones including a date like "There is not any appointment available on Tuesday the 10th". These templates were completed and synthesised on the fly. Speech synthesis was performed by an Orange internal synthesiser. The synthesiser allowed to tweak the intonation and pronunciation of the fully static prompts. This property was exploited and the resulting prompts were stored in audio files called by the system when needed.

5.2.3 Dialogue Management

Issues Previously Identified

Similarly to what was demonstrated by Dybkjaer et al. [1996], many of the problems detailed in the CLASSiC evaluation [Laroche et al., 2011] could be explained in terms of uncooperative behaviour of the system. For instance, it was noticed that users were sometimes confused by system feedbacks. Let us take the example of a user saying she would like to book an appointment on Friday afternoon. In this case, most of the time, the user meant the upcoming Friday afternoon. Yet, in accordance with Grice's quality maxim [Grice, 1989], systems 3 and 4 would not make any assumption on the desired week. Thus, if the first appointment available was Friday afternoon of the following week, both systems 3 and 4 would have directly proposed this appointment without stipulating that the upcoming Friday was not available and thus, would have violated the quantity maxim. Users tended to distrust speech recognition so, in this case, they thought the system had not understood what they said and they often chose to refuse the proposition and repeat their request. NASTIA disambiguates these cases by prompting the user with an implicit confirmation. To such a user utterance, NASTIA answers *Friday the 16th* to let the user know that it was supposing she was meaning the upcoming Friday. This new formulation respects Grice's quantity maxim as it provides the user with the necessary amount of information for her to understand the course of the dialogue.

Other modifications of the same nature were made resulting in many prompts being reformulated to move towards a better accordance with the Gricean cooperativity principles and make the system less of a black box for the user.

Dialogue modelling

Appointment scheduling is modelled as a slot-filling task with three parameters: day, week, and half-day (morning or afternoon). Following the design paradigm of Disserto, NASTIA's dialogue manager is based on

[1] www.nuance.com

a finite state machine. Each node of the machine is a dialogue phase. Dialogue phases in NASTIA are for example: Welcome, Confirm, Ask_open_Question, Ask_For_Day, Recovery (from speech recognition rejection or user time out), etc.

RL was integrated into this automaton with the MVDP hybrid framework (see Chapter 2). Following this framework, a dialogue phase may contain one or several module(s). NASTIA contains five modules in five different dialogue phases.

The first module determines the negotiation strategy. The User Initiative (UI) strategy consists of asking the user: "When would you like to book an appointment ?". System Initiative (SI) asks the user for the day, week, and half-day during three different dialogue turns. The order of the questions is decided as it was decided in System 4. In addition to these classical strategies, a third option was implemented where the system directly proposes a List of Availabilities (LA) to the user, waiting for her to interrupt the list after an adequate appointment has been proposed. This last option was inspired by recent work on incremental dialogue management (see Chapter 2).

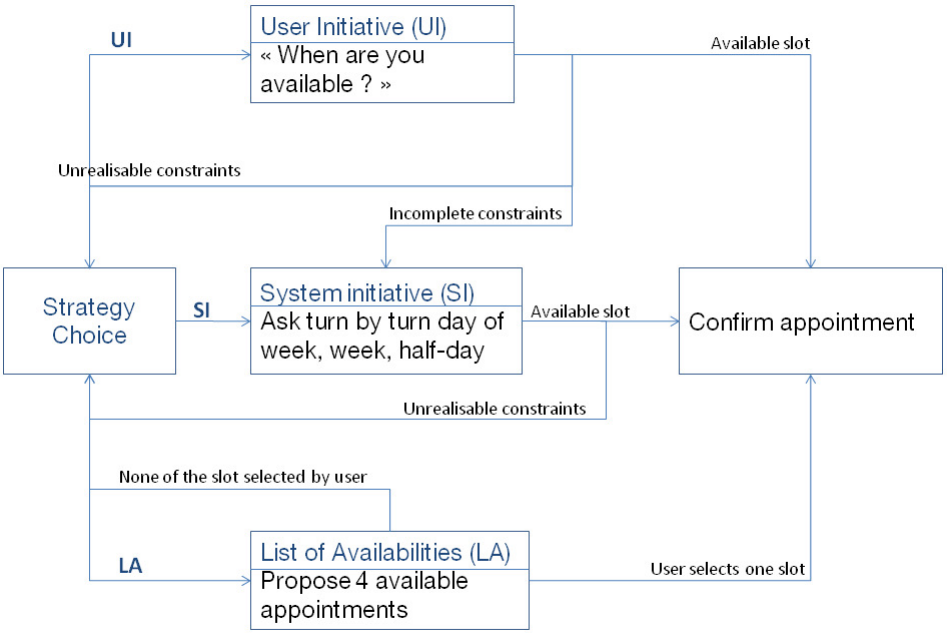


Figure 5.2: NASTIA's appointment scheduling strategies.

Figure 5.2 describes the way NASTIA carries a negotiation to set an appointment. The system chooses which strategy to follow at the beginning of each dialogue and after each appointment setting failure. This leaves NASTIA the opportunity to adapt its way of realising the task in function of the course of the dialogue as it was proposed for instance by [Chu-Carroll, 2000] and [Litman and Pan, 2002]. If the system picks out the listing strategy, four available slots are proposed to the user. If the user has not interrupted the system after the fourth proposition, the system asks the user to confirm that none of the slots is suitable and then the

Initiative strategy	User Initiative (UI); System Initiative (SI); List of Availabilities (LA)
ASR rejections; User inactivity	Play a help message; Tell the user her utterance was not understood
Confirmation strategy	Explicit confirmation; Implicit confirmation; No confirmation
System calendar information	Give information; Do not give information
Help message	Recall dialogue context; Give the possibility to cancel + Recall dialogue context + Recall available commands; Give the possibility to cancel + Recall dialogue context

Table 5.1: Actions of the appointment setting system.

negotiation strategy is decided upon again: the system may either keep going with this strategy or switch to user initiative or system initiative. While the system lists its availabilities, the user can also interrupt the system to propose some constraints. For instance, if the system starts listing slots for a week during which the user is not available, the user has the possibility to interrupt the system and say "next week". If so, the system switches to system initiative and asks for the missing slots.

The second module concerns contextual help generation. The user may express a help request at any moment of the dialogue. If so, the system may combine three components of help messages:

- (a) Tell the user: "You have required the help section" and leave her the possibility to answer "no" in case the system misunderstood the request.
- (b) Recall the current context of the dialogue (e.g., "You were asked when you would like to make an appointment") and tell the user what she can say (e.g., "You can answer by saying for instance *this Friday afternoon*, *this week in the morning* or *Monday the 19th*.")
- (c) Recall the available commands ("Repeat" and "Help").

NASTIA chooses amongst three combinations: (b), (a) + (b) + (c), or (a) + (b).

The third module is visited after a user has proposed a time slot. NASTIA chooses between three confirmation strategies. Following the first strategy, the system does not ask for any confirmation. The implicit confirmation strategy simply consists of repeating what was understood. In case the system misunderstood her utterance, the user can barge in to correct the system. The explicit strategy requires a yes/no answer. The system asks: "I understood you were available on [understood date]. Is it correct?".

The fourth module was implemented to compare two strategies for speech recognition rejections and user time-out recovery. The SDS may play the (b) help message or inform the user that she was not understood/heard so that the user repeats/says something.

Like CLASSiC System 2, NASTIA can provide information about its calendar after an appointment setting failure or after the user has expressed some constraints. This is decided by the fifth module. System 2 could tell the user that there were no appointments available except x and y given the user's constraints. During a dialogue, NASTIA keeps up to date the number of available and unavailable slots matching user constraints. If one of the two numbers is below or equal to three, the module can decide to list the available/unavailable slots. These slots might be completely defined but they might also simply be days, half-days, or weeks. For instance, if a user says she is available this week during the morning, the system may answer "This week, during the morning, Tuesday and Friday are not available". All of NASTIA's modules as well as their action

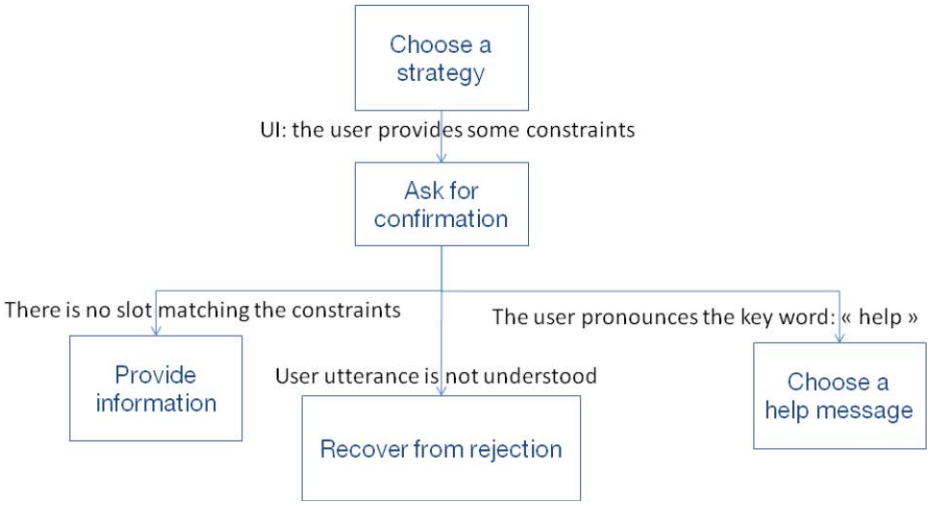


Figure 5.3: Example of dialogue triggering all of NASTIA's modules.

sets are gathered in Table 5.1. As in System 3, the sets of local contexts associated with each module are empty. The information states are thus entirely defined by the modules. Figure 5.3 illustrates the dialogue events triggering the visit to the system's modules. The following section describes the collection of a corpus of evaluated dialogues with NASTIA and their annotation with KPI.

5.3 DINASTI

5.3.1 Recruitment

All volunteers for the experiment were Orange employees recruited by Email. The first recruitment campaign received 627 answers. An Email was then sent to these subscribers with 5 hyperlinks. Each hyperlink was associated to a code to make sure each call was unique. A code was composed of the call identifier (5 digits) and the scenario number (2 digits, see in Chapter 4 the description of CLASSiC System 3). A last digit was added for Cyclic Redundancy Check (CRC).

A user guide was attached to the Email. It explained the scenario, how to make a call, and then fill in the questionnaire. The guide is reproduced in Appendix B. After clicking on one of the links, the user was sent to a web page explaining the scenario which was the following:

Today is Monday, July 12th and your landline is not functional. After it diagnosed that the intervention of an engineer on site was required, the technical service has redirected you to a spoken dialogue system to book an appointment. Your aim is to set an appointment at one of the available slots on the following calendar.

Then the user was displayed a calendar as the one shown on Figure 4.3 in Chapter 4. Under the calendar was indicated the phone number to call. This phone number connected to a DTMF front-end system that asked the user to enter the code. The front-end system then transmitted the call to NASTIA after having extracted from the code the following information: the call identifier to be written in the logs and the scenario number so that NASTIA could download the corresponding calendar. The exact same calendars and scenarios as the ones that

were used during the CLASSiC experiment were used, in order to be able to compare the different approaches. After performing the call, users filled in an evaluation questionnaire on the same page as the one where the calendar was displayed.

In total, 385 participants made 1 to 5 calls, with an average of 4.6 calls per participant. This resulted in 1734 dialogues and 21587 system-user exchanges, among which 7508 were decision turns, i.e., turns where the system needed to choose amongst several actions. In accordance with the methodology described in Figure 5.1, during corpus collection, the system followed a pure exploratory policy, i.e., randomly picked an action at each decision turn.

5.3.2 Evaluation

Evaluation Questionnaire

The evaluation questionnaire was the following:

1. Did you book an appointment?
2. Was the appointment booked for one of your available slots?
3. When did you book the appointment?
4. During your dialogue with the system, you knew what to say.
5. You could easily recover from system misunderstandings.
6. Understanding the system was easy.
7. The system provided enough information for the dialogue to be easy to follow.
8. The dialogue with the system was efficient.
9. The dialogue with the system was fluid.
10. The system was concise.
11. Overall evaluation.
12. Do you have any remarks or comments?

Questions 1 and 2 required a yes/no answer. For Question 3, the user had to select the appointment date if an appointment had been set. Questions 4 to 10 were evaluated according to a six-point Likert scale: completely disagree, disagree, mostly disagree, mostly agree, agree, completely agree. Another option was added to Question 5 in case there had been no speech recognition mistakes. For Question 11, the users were asked to rate the dialogue on a scale of 1 to 10. Finally, Question 12 was free text, to report any problem or give a general opinion on the system.

Questions 4 to 9 were adapted from the SASSI and PARADISE questionnaires (see Chapter 2). Question 10 was added to assess the fact that the system did not provide more information than required (Grice's maxim of quantity).

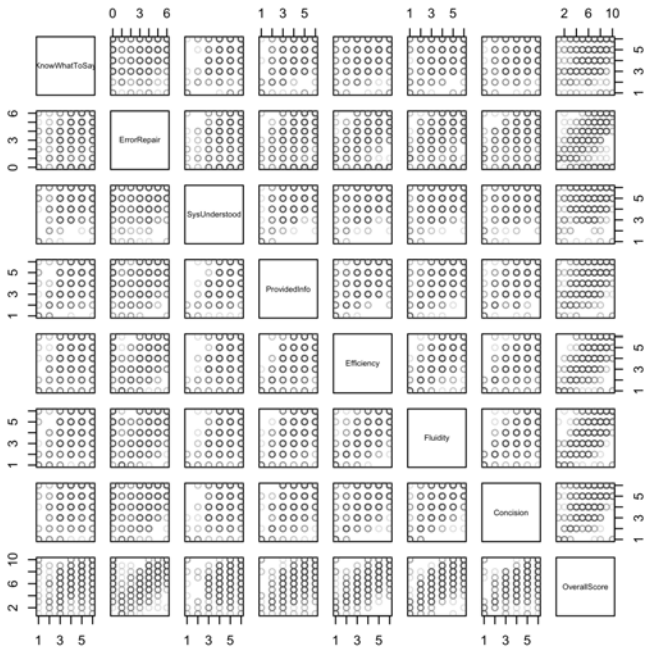
Questionnaire Analysis

The questionnaire for the evaluation of NASTIA is different than the one that was used for the CLASSiC systems (see the previous chapter). The CLASSiC questionnaire was similar to the PARADISE questionnaire (see Chapter 2). As said in Chapter 2, it was chosen in this work to target the evaluation towards dialogue management and thus unrelated questions from the PARADISE questionnaire such as the one about the system's voice were avoided. Figure 5.4 shows the correlations between the answers and the questionnaires' items for NASTIA and System 3. For NASTIA, there is a visible correlation between the overall evaluation and the items related to error-repair easiness, efficiency, and fluidity. As shown in Figure 5.5, there is a strong correlation between the evaluation of NASTIA's efficiency and the easiness to repair from ASR/NLU errors. Note that the error-repair item took the null value only when there was no ASR/NLU mistake during the dialogue, hence the high efficiency value for this case. It is coherent that the ability to recover from error easily entails more efficient dialogues. A multivariate linear regression on the centered and scaled data showed that the two most significant parameters were efficiency and fluidity. Therefore, the following analysis will focus on these two items. In the case of System 3, overall evaluation is mostly correlated with task ease and future use. Multivariate linear regression on these items returns:

$$\begin{aligned} &\text{overall\_evaluation (NASTIA)} = 1.43 + 0.71\ \text{efficiency} + 0.58\ \text{fluidity}, \\ &\text{overall\_evaluation (System 3)} = 2.75 + 0.67\ \text{future\_use} + 0.35\ \text{task\_ease}. \end{aligned} \tag{5.1}$$

The highest weights correspond to efficiency for NASTIA and future use for System 3. The correlation between overall evaluation and efficiency/future use is represented in Figure 5.6. The patterns are similar for the two systems. Future use measures whether or not the user is likely to use this kind of system in the future, i.e. if the system's efficiency is competitive compared to existing solutions which are, in the case of appointment setting, dialogues with human operators. Future use is thus a measure of efficiency. It is coherent that in the two cases, the efficiency is the best explanatory feature for overall evaluation. It can be concluded from this study that even if the systems were evaluated with different questionnaires, they can be compared in terms of overall evaluation. Such a comparison will be performed in the following section. Beforehand, the other items in NASTIA's evaluation questionnaire are discussed. Figure 5.7 shows the correlation between the overall evaluation score and the other items in the questionnaire. A common pattern can be identified for the overall scores 8, 9, and 10. Indeed, they correspond to strong agreements with the questionnaire's items (75% of the data between 5 and 6 which correspond to *agree* and *completely agree*). In addition, a significant correlation can be observed between the scores and the items evaluating the system's concision and the amount of information provided by the service. The item that seems to be the least correlated with the overall score is the one evaluating the user's ability to understand the system. This item is more linked to NLG and speech synthesis than to dialogue management. During the experiment, users reported that the audio quality was not very good. Nevertheless, this does not seem to have had a significant impact on the overall score.

As a conclusion, the overall score can be significantly explained by the other items in the questionnaire, which were meant to measure dialogue management quality. The adjusted R^2 for the reported multivariate linear regression is 0.63 and the p-value is under 10^{-16}. As a consequence, the overall score can be seen as a relevant measure of the quality of dialogue management and it makes sense to derive from this metric a reward function for the dialogue manager.



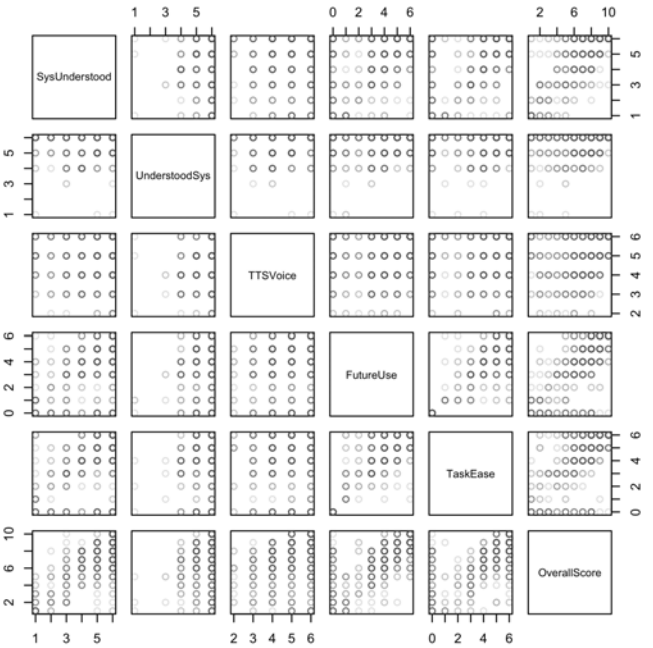


Figure 5.4: Correlations between the answers to the evaluation questionnaires for NASTIA (left) and System 3 (right).

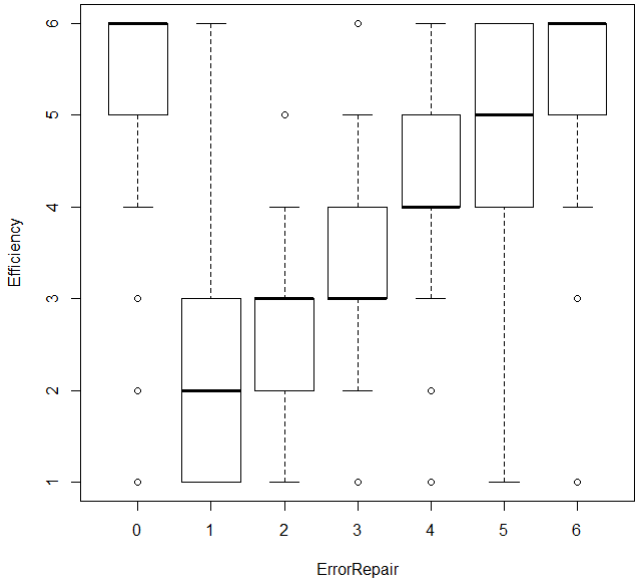


Figure 5.5: Boxplot representing the correlation between NASTIA's perceived efficiency and the easiness to repair from ASR/NLU errors.

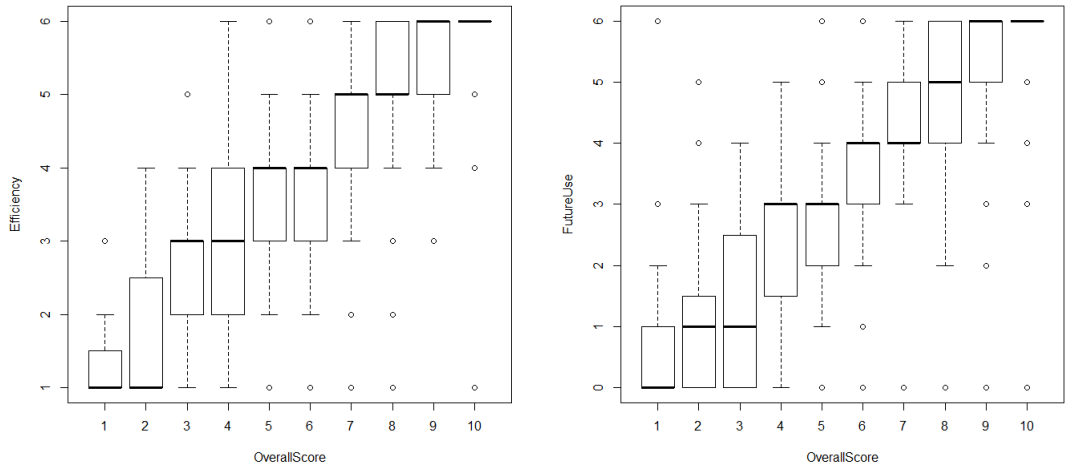


Figure 5.6: Correlations between overall evaluation and efficiency for NASTIA (left), overall evaluation and future use for System 3 (right).

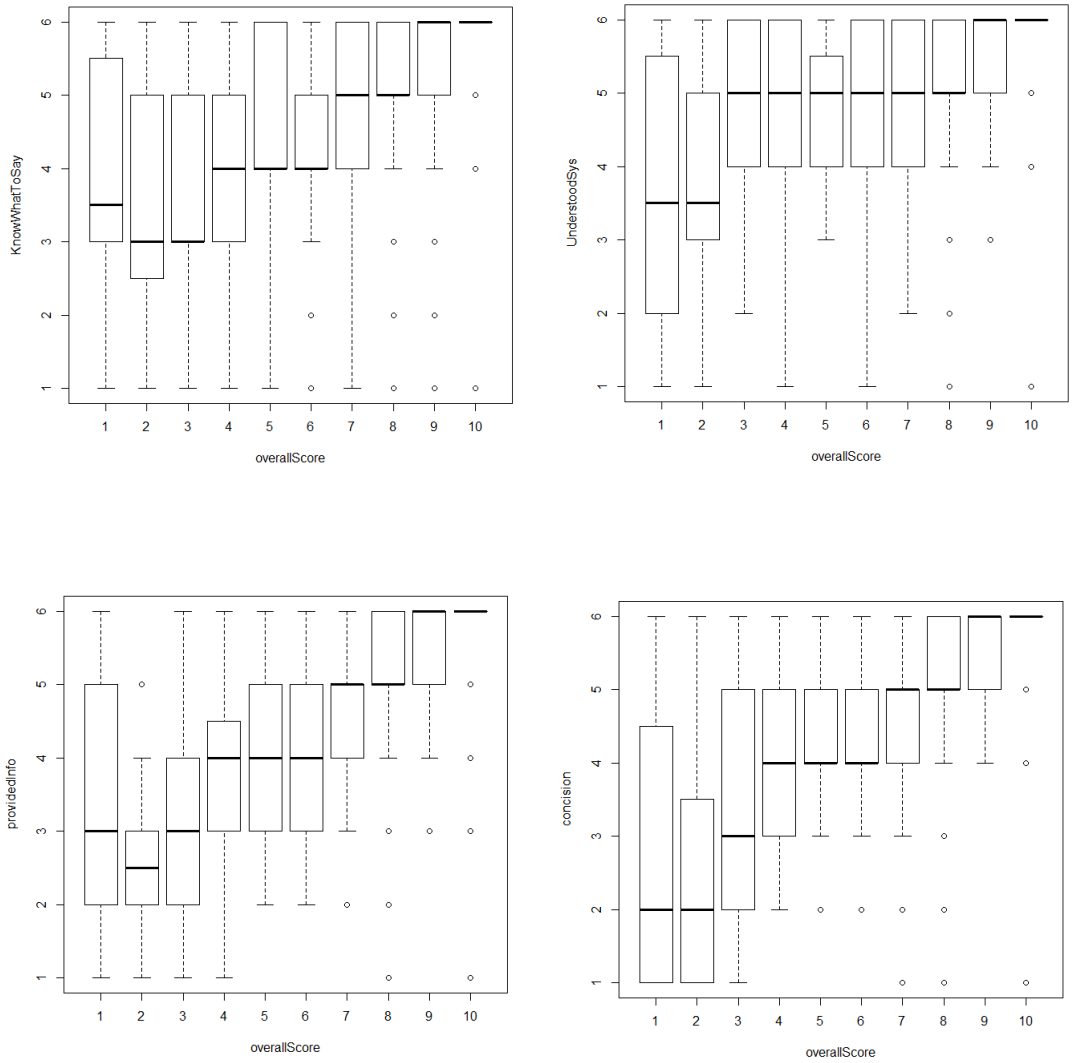


Figure 5.7: Correlation between NASTIA's overall score and, from top to bottom, left to right: the fact that the user knew what to say, the easiness to understand the system, the fact that the system provided enough information to the user, the system's concision.

System	STC	UTC	Duration (sec)	Rating	Number of calls
System 2	$79 \pm 3\%$	$68 \pm 4\%$	97 ± 5	5.21 ± 0.23	628
System 3	$81 \pm 3\%$	$83 \pm 4\%$	69 ± 3	7.40 ± 0.17	740
System 4	$83 \pm 3\%$	$85 \pm 3\%$	98 ± 5	6.54 ± 0.18	709
NASTIA	$88 \pm 2\%$	$92 \pm 1\%$	75 ± 3	7.75 ± 0.09	1734
NASTIA UI	$87 \pm 3\%$	$89 \pm 3\%$	84 ± 6	7.57 ± 0.16	587
NASTIA LA	$92 \pm 3\%$	$95 \pm 2\%$	61 ± 5	8.28 ± 0.14	562
NASTIA SI	$87 \pm 3\%$	$92 \pm 2\%$	79 ± 5	7.43 ± 0.17	585

Table 5.2: Performance comparison between NASTIA and CLASSiC's systems 2, 3 and 4. STC is System Task Completion and UTC, User Task Completion. Time is measured in seconds. 95% confidence intervals are provided for the mean of the binomial (STC and UTC) and the normal law (Time and Rating).

Results and comparison with the CLASSiC systems

The ratings for the *overall evaluation* item in the questionnaire are compared with the ones obtained with CLASSiC's systems 2, 3 and 4. The comparison of the systems' performance is also made on the basis of KPI such as System and User Task Completion (resp. STC and UTC) and dialogue duration (in seconds). These results are given in Table 5.2. User task completion is derived from the answers to questions 1 and 2 in the questionnaire (questions 1 and 2 were the same for NASTIA's evaluation and the CLASSiC systems' evaluations, see the previous chapter). System task completion is equal to 1 if the right appointment has been booked and 0 otherwise.

There were 628 evaluated dialogues for System 2, 740 for System 3, and 709 for System 4. Systems 3 and 4 shared the same automatic speech recognition, natural language understanding, and text-to-speech components as NASTIA. During the CLASSiC experiment, these systems largely outdid System 2 concerning STC and UTC. System 3 was the one that did best in terms of overall evaluation and led to the shortest dialogues [Laroche et al., 2011].

NASTIA performed similarly to System 3 in terms of overall evaluation, although dialogues were in average 6 seconds longer with NASTIA. STC and UTC are significantly higher. In table 5.2 are also given 95% confidence intervals for the KPI in function of the first decision made by the system, that is to say the system's first negotiation strategy. Listing availabilities entailed significantly higher evaluations and shorter dialogues, no matter the policy followed by the system afterwards. Dialogues are shorter and the mean evaluation with listing availabilities is clearly higher than the mean rating of CLASSiC's System 3.

The main difference between NASTIA and systems 3 and 4 concerns the negotiation strategy. NASTIA can try several strategies during the same dialogue. In addition to this flexibility, the ability for the user to barge in after an ASR rejection was well-perceived. In case of speech recognition rejection or user time out, System 3 would tell the user that she was not understood/heard and it would then repeat its latest utterance. As for System 4, it would ask the user to confirm its first then second hypothesis and if neither was accepted by the user, the system would repeat its latest question. It was observed during the CLASSiC experiments that users tended to try and barge in instead of waiting for the system to repeat its question [Laroche et al., 2011]. NASTIA leaves this possibility to the user. Finally, providing information to the user about the system's availabilities given the user's constraints is in better accordance with Grice's principles of cooperativity since the system contributes to the dialogue by providing as much information as it can according to its current

beliefs.

Remarks and Comments on the System

The answers to question 12 in the evaluation questionnaire shed light on the users' current perception of task-oriented automated spoken dialogue.

First, several users expressed the fact that they would have appreciated to be more guided during the dialogue. The approach in NASTIA is to let the user barge in at almost any moment of the dialogue. Thus, there are blanks of a few seconds after system utterances. For instance, after the system has told the user that no appointment matches her constraints, it waits in case the user directly specifies new constraints. Users wrote that they did not know what to say during these blanks and they would have rather the SDS took over the dialogue more quickly. Testers are more accustomed, especially in a commercial context, to hear the system say "your turn to talk" when they are supposed to interact. On the contrary, some users preferred the user initiative strategy to the more directive listing one. Nevertheless, after the system has told that an appointment was not available, they would have wanted the system to switch strategy, to be more directive. This shows that users might not be completely ready for natural dialogue, in the sense of human-like dialogue, with a task-oriented SDS. Contrary to listening-oriented systems [Meguro et al., 2009], task-oriented systems are still expected to be more directive and allow a narrower range of user utterances.

Another interesting point about the user initiative strategy is that users progressively learnt to use it. As said before, testers interacted at most five times with the system. Some users were confronted to user initiative more than once. They wrote that at first, they were not sure of the date format expected by the system but then they found out that the *day of week/day of month/half-day* format was well understood by the system and enabled to speed up ruling out slots. Thus, it seems important to keep the three negotiation strategies as, the more users call this system, the more comfortable they are with user initiative but the other two strategies are important to keep for less experimented users. The choice of strategy according to current dialogue context should be successfully learnt with reinforcement learning. If a strategy fails, dialogue history and what was observed during previous dialogues should inform NASTIA about what strategy to try next. This confirms that dialogue management requires a finer representation of the course of the dialogue than the one only relying on context-free modules. To make an efficient decision, the system needs to know more then its current module, it must also take into account dialogue history.

5.3.3 Corpus annotation

Corpus annotation was performed on the basis of the parameters described by Schmitt et al. [2008]. This feature set is composed of features returned by the speech recognition, natural language understanding, and dialogue management modules. The features were shown to be relevant to predict the interaction quality with an SDS [Schmitt et al., 2011, Ultes and Minker, 2013] and to identify problematic dialogues [Walker et al., 2002]. The feature set is described in Table C.1 in Appendix C. In the same appendix, the system and user dialogue acts are described in Tables C.2 and C.3.

The #RuleUsage and #TagUsage features are returned by the system's Natural Language Understanding (NLU) component. As said earlier, NLU in NASTIA is rule-based: 38 grammar rules can be triggered to understand the user's utterance. ASR and NLU only operate on a restricted domain defined by the tags of the #TagUsage feature.

5.3.4 Corpus usage

The DINASTI corpus can be used for multiple purposes. As said in the introduction of this thesis, the corpus will be used to illustrate the feature selection performed by the algorithm proposed for state space representation and its interpretability on a large feature space. However, other research, for instance, research on user simulation [Schatzmann et al., 2006, Pietquin et al., 2009, Chandramohan et al., 2011] could be carried on the corpus. The negotiation task implies unusual constraints on user simulation design. Indeed, it is not a slot-filling task with a static goal. In DINASTI, the user's goal might change during the dialogue, when an appointment is unavailable. Besides, the user may take over the task or dialogue initiative at any point of the dialogue, which is interesting for user adaptivity and expertise modelling research. In order to encourage research on these three topics, DINASTI will be soon made available online.

5.4 The Appointment Scheduling Simulator

5.4.1 Dialogue Modelling

The simulator is designed as the following MDP:
- $S = \{$REPEAT, REFUSE_AND_ASK, NEGATE_AND_ASK, ACCEPT, CONFIRM$\}$
- $A = \{$EXPLICIT_CONFIRM, REFUSE_AND_ASK, ACCEPT, REJECT$\}$
- P is unknown.
- R is unknown.
- $\gamma = 0.99$.

As said in the introduction, the simulator illustrates a different setting than NASTIA's. In this case, there is not only one slot which is free for both the user and the system. Instead, N slots are available. Each slot n_i is associated with a preference score p_i. The user and the system have different preference scores. It is supposed that both the system and the user can rank the different time slots in order of preference. To model this, a preference p_i equal to $\frac{i}{\text{total number of slots}}$ is associated to the ith slot n_i. For instance, if the total number of slots is 5, the preferences would be 0.2, 0.4, 0.6, 0.8 and 1. The total number of slots was set to be equal to 15. New dates and preference scores were drawn at the beginning of each dialogue.

Each dialogue starts with a proposition by the system (ASK). Then, during the dialogue, the system can either refuse a slot and propose a new one (REFUSE_AND_ASK), ask for an explicit confirmation (EXPLICIT_CONFIRM), accept a proposition (ACCEPT), reject the utterance returned by ASR (REJECT), or ask the user for a missing item in the date of the appointment (day, day of the month, or hour, ASK_MISSING). This last action is only used when the ASR deletes one of the element said by the user. The user always gives complete slots in the form day / day of the month / hour, for instance *Friday 11th at 8 o'clock*. However, the ASR module might only understand two of the elements, for instance *Friday 11th*, in which case the system will say *Please specify an hour for the appointment*. The actions ASK and ASK_MISSING are not part of the actions of the MDP modelling the system's behaviour because these actions are not learnt via RL, ASK is always and only used at the beginning of each dialogue and ASK_MISSING is only summoned when the ASR returns an incomplete date. The states of the MDP correspond to UDA: at each time step of the dialogue, the informational component known by the system is the latest UDA.

The user shares several actions in common with the system, namely REFUSE_AND_ASK and ACCEPT. The REPEAT action is summoned when the system has told the user that her utterance had not been understood (REJECT). When the system asks for a confirmation but did not understand correctly what the user said, the

Speaker	Dialogue Act	Utterance
System	ASK	I propose: Wednesday 7 at 20 o'clock.
User	REFUSE_AND_ASK	No, I propose: Monday 19 at 16 o'clock.
System	REFUSE_AND_ASK	No, I propose: Wednesday 21 at 16 o'clock.
User	ACCEPT	Ok.
System	ASK	I propose: Tuesday 25 at 12 o'clock.
User	ACCEPT	Ok.
System	ASK	I propose: Monday 10 at 17 o'clock.
User	REFUSE_AND_ASK	No, I propose: Monday 21 at 23 o'clock.
System	EXPLICIT_CONFIRM	Did you say: Wednesday 21 at 23 o'clock? Please answer yes or no.
User	NEGATE_AND_ASK	No, Monday 21 at 23 o'clock.
System	REFUSE_AND_ASK	No, I propose: Wednesday 1 at 13 o'clock.
User	REFUSE_AND_ASK	No, I propose: Thursday 1 at 19 o'clock.
System	ASK_MISSING	Please specify an hour for the appointment.
User	REPEAT	Thursday 1 at 19 o'clock.
System	REJECT	I did not understand what you just said.
User	REPEAT	Thursday 1 at 19 o'clock.
System	REJECT	I did not understand what you just said.
User	REPEAT	Thursday 1 at 19 o'clock.
System	EXPLICIT_CONFIRM	Did you say: Thursday 1 at 19 o'clock? Please answer yes or no.
User	CONFIRM	Yes.
System	ACCEPT	Ok.

Table 5.3: Examples of dialogues with the appointment scheduling simulator. Actions of the system and the user are uniformly random.

user negates and then repeats the correct utterance (NEGATE_AND_ASK). Finally, if the system is right about the utterance, the user confirms it (CONFIRM). Table 5.3 presents examples of dialogues generated with random behaviours for both the system and the user.

5.4.2 Automatic Speech Recognition Simulation

ASR confidence scores are simulated with a word error rate of 20%. A word substitution happens with probability 0.7, a suppression or addition, each with probability 0.15. The score is computed by drawing from a normal distribution centered in -1 and with a variance of 1 in the case of a misrecognition (centered in 1 otherwise) and by then applying the sigmoid function to this value.

5.4.3 Modelling User Satisfaction and Scoring the Dialogues

The second step of the methodology in Figure 5.1 is to collect rated dialogues with the system following a purely exploratory policy. As said in the introduction, the simulator's task is not only to optimise user

satisfaction but also to maximise the preference score of the appointment. Therefore the scoring function has the following form:

$$\text{score} = \text{preference} + \alpha \times \text{user\_satisfaction} \tag{5.2}$$

where α is between 0 and 1. If α equals 1 than the simulator must maximise jointly the preference score of the appointment and user satisfaction. If $\alpha < 1$ then more importance is given to the preference score and if α equals 0, user satisfaction is not taken into account at all. In all that follows, results with the simulator will be given with α set to 1. Equation 5.2 necessitates an estimator of user satisfaction. It was chosen to train an estimator on the DINASTI corpus with generic dialogue parameters. Each dialogue is represented by the following statistics: dialogue duration (in dialogue turns), number of rejections from speech recognition, average ASR confidence score and user task completion. User task completion is computed as follows: if the booked appointment has a preference score superior or equal to the fifth highest preference score (0.7 for the total number of slots equal to 15), then the task is considered to have been completed and task completion is set to 1. Otherwise, task completion is set to 0. The estimator of user satisfaction is Support Vector Ordinal Regression (SVOR, [Chu and Keerthi, 2007]) trained on the DINASTI corpus. Details about this estimator are given in Chapter 9. It takes as input a vector composed of the four dialogue parameters previously listed and returns a score between 1 and 10. This score is scaled and centered in order to have the same order of magnitude as the preference score.

5.4.4 User Simulation

In this setting, the goal of the user is to maximise her own satisfaction. It is reasonable to consider the user as a decision maker who is trying to optimize a certain objective function and thus to model the user's decision process as an MDP [Pietquin and Hastie, 2010, Chandramohan et al., 2011, 2012]. This is the approach chosen for these simulations. Chandramohan et al. [2011] proposed to model the user as an MDP and then learn its behaviour through Inverse Reinforcement Learning (IRL, [Russell, 1998, Ng and Russell, 2000]). IRL considers the problem of defining a reward function for an MDP without one from examples of expert or quasi-expert trajectories [Abbeel and Ng, 2004, Ziebart et al., 2008, Klein et al., 2011, 2012]. Chandramohan et al. showed that it was possible to learn efficiently a reward function from a set of dialogues.

For the simulator, it was chosen to model the user's goal as the maximisation of her satisfaction and then make the user negotiate with herself to learn which dialogue strategy enables to maximise her satisfaction. User satisfaction was here also computed with SVOR. However, if the user negotiates with herself, there are not problems related to speech recognition so another SVOR estimator was trained only on dialogue duration and user task completion. Teaching an RL agent to play a game by having the agent play against itself has been proven to be an efficient way of exploring interesting strategies and has lead to successful strategies for games like Backgammon [Tesauro, 1995]. It is well-suited for dialogues involving negotiation like the dialogues for appointment scheduling simulated here. In this context, the user has to learn when to accept a slot and when to reject one based on dialogue duration and task completion. Formally, the user is modelled as follows:
- $S = \{$REFUSE_AND_ASK, ASK$\}$.
- $A = \{$REFUSE_AND_ASK, ACCEPT$\}$.
- P is unknown.
- $R =$ user_satisfaction (estimated with SVOR trained on DINASTI with dialogue duration and task completion for parameters).
- $\gamma = 0.99$.

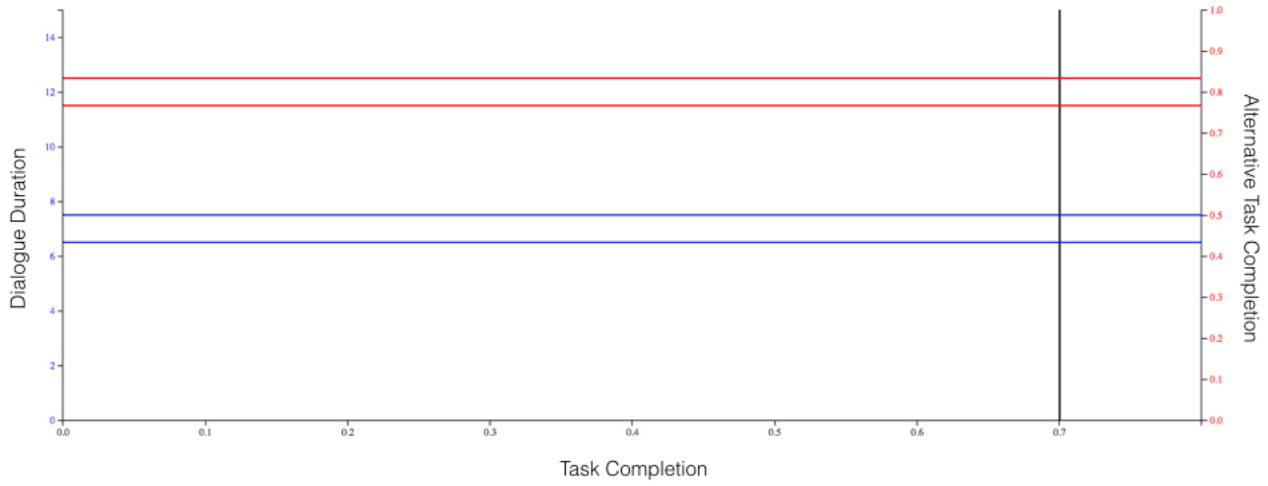


Figure 5.8: Grid representing the state space corresponding to the UDA REFUSE_AND_ASK.

Each dialogue starts with a proposition (ASK) and then the user can accept or reject and suggest another slot. The state space modelling the user is not sufficiently informative to correctly learn when to accept or reject an appointment. Indeed, since user satisfaction depends on dialogue duration and slot preference, these two parameters should be included into the state space. A third parameter is also added. This parameter is the value of the slot which would be proposed by the user if she refused the slot which has been suggested to her. This parameter will be named *task completion alternative*. In order to make the state space take into account these parameters, it is proposed to represent the state space as a grid: for each UDA (REFUSE_AND_ASK and ASK), a 3D grid discretising dialogue duration, task completion, and alternative task completion is learnt. The grids were learnt by simulating 100000 dialogues with a random policy for the user and then applying entropy-based binning to discretise task completion, alternative task completion, and dialogue duration [Fayyad and Irani, 1993, Rieser and Lemon, 2011]. The grid learnt with this technique on the 100000 dialogues and corresponding to REFUSE_AND_ASK is represented in Figure 5.8. The discretisation produced 24 states for the REFUSE_AND_ASK UDA and 2 states for the ASK UDA. The UDA ASK is only visited after the first turn of each dialogue so the duration is always equal to 1. The two states computed for ASK are: task completion < 0.7 and task completion ≥ 0.7. Based on this representation, user behaviour was learnt by making the user negotiate with herself. To learn a policy, the algorithm known as SARSA(λ) [Sutton and Barto, 1998] was used. SARSA(λ) is an example of model-free RL algorithm. As many other RL algorithms, it can be framed as the alternation of two processes: policy evaluation and policy improvement. The learner starts with a policy π_0 (which might be random or an implementation of prior domain knowledge). The first step is to evaluate this policy. Then, according to the resulting Q-function, the policy is improved. This alternation goes on until a stable policy is found. As explained in Chapter 2, model-free RL algorithms do not rely on an estimation of the transition probabilities P. SARSA(λ) is an instance of *Temporal Difference* (TD) learning where the value of a state-action pair is updated according to the estimated values of other pairs. The policy is approximated

based on the fact that

$$\begin{aligned} Q^{\pi}(s,a) &= E_{\pi}\left[r_t \mid s_t = s, a_t = a\right] \\ &= E_{\pi}\left[R_{t+1} + \sum_{t' \geq 1} \gamma^{t'} R_{t+t'+1} \mid s_t = s, a_t = a\right] \\ &= E_{\pi, s_{t+1} \sim P_{a_t}(s_t,.)}\left[R_{t+1} + \gamma Q^{\pi}(s_{t+1}, a_{t+1}) \mid s_t = s, a_t = a\right]. \end{aligned}$$

Policy evaluation for SARSA is done through the following update rule, for each transition $(s_t, a_t, s_{t+1}, a_{t+1}, R_{t+1})$:

$$Q^{\pi}(s_t, a_t) \leftarrow Q^{\pi}(s_t, a_t) + \alpha\left[R_{t+1} + \gamma Q^{\pi}(s_{t+1}, a_{t+1}) - Q^{\pi}(s_t, a_t)\right]. \tag{5.3}$$

At each time step, the current estimation of $Q^{\pi}(s_t, a_t)$ is corrected by the discounted residual between $R_{t+1} + \gamma Q^{\pi}(s_{t+1}, a_{t+1})$ and $Q^{\pi}(s_t, a_t)$. The learning factor α is a real value between 0 and 1. If this factor is equal to 0, then the estimation of Q would never take into account new information whereas a factor of 1 would only consider the new information $R_{t+1} + \gamma Q^{\pi}(s_{t+1}, a_{t+1})$. Based on the estimation of the $Q-$function, at each time t, it is possible to chose which action a_t to perform either by choosing the one that maximises the Q-function at s_t or by choosing to explore the state space by choosing another action according to an exploration-exploitation strategy (see Chapter 2). SARSA(λ) is different than SARSA in that it leverages *eligibility traces*, an RL mechanism that behaves like a short-term memory. As shown in Equation 5.3, the update of $Q^{\pi}(s_t, a_t)$ only depends on the immediate reward R_{t+1} and the Q-value of the next state-action pair $Q^{\pi}(s_{t+1}, a_{t+1})$. The idea of eligibility traces is to extend the update to later rewards and Q-values. When a state-action pair (s, a) is visited, its eligibility trace is set to 1. Then, at each time step, this trace is decreased by a factor $\gamma \times \lambda$: $e_t(s, a) = \gamma\lambda e_{t-1}$. The pair (s, a) is updated at each time step, as long as its eligibility trace is not null. Thus, if (s, a) is visited at time t, its Q-value is update according to R_{t+1} and $Q^{\pi}(s_{t+1}, a_{t+1})$ but also R_{t+2} and $Q^{\pi}(s_{t+2}, a_{t+2})$, etc. For each $t + k$, such that $e_{t+k}(s, a)$ is not null:

$$Q^{\pi}(s, a) \leftarrow Q^{\pi}(s, a) + \alpha e_{t+k}(s, a)\left[R_{t+k} + \gamma Q^{\pi}(s_{t+k}, a_{t+k}) - Q^{\pi}(s_{t+k-1}, a_{t+k-1})\right]. \tag{5.4}$$

For the exploitation-exploration strategy, it was chosen here to follow an ϵ-greedy strategy with ϵ equal to 0.1 (which means that with probability 0.9, the optimal action was chosen and with probability 0.1, a random action was chosen). The learning factor α was set to 0.1 and λ was set to 0.9. To speed-up learning, the Q-values were initialised at 10 (the mean performance with a random policy is 9.3). The result of learning is shown on Figures 5.4.4 and 5.4.4. Figure 5.4.4 shows that the user successfully learns to maximise her own satisfaction which starts at 9.3 with a random behaviour and reaches around 9.8 with the behaviour learnt with SARSA(λ). Figure 5.4.4 shows that the learnt behaviour is better in terms of user satisfaction in that it maximises task completion while keeping a relatively low dialogue duration.

On top of the policy learnt for the user behaviour, in order to model user's impatience, if the dialogue reaches 30 dialogue turns, the user hangs up and the dialogue is a failure, rewarded with 0.

5.4.5 Corpus Collection and Annotation

Following the methodology, a corpus of dialogues with a random policy should be gathered and annotated with KPI. 100000 such dialogues were generated with the user behaviour learnt in the previous section. Each dialogue was annotated on a turn-level basis with the following KPI: number of turns, mean ASR score, number

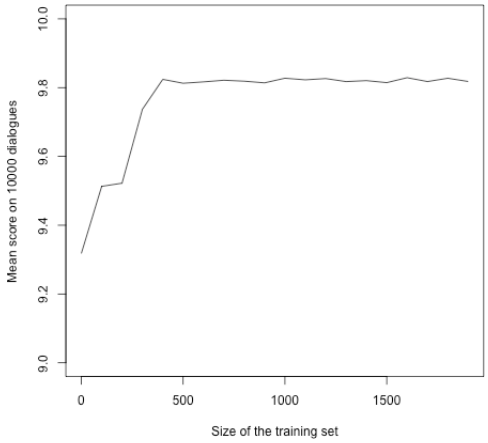


Figure 5.9: Result of the learning with SARSA(λ) of a user behaviour for the appointment scheduling simulator.

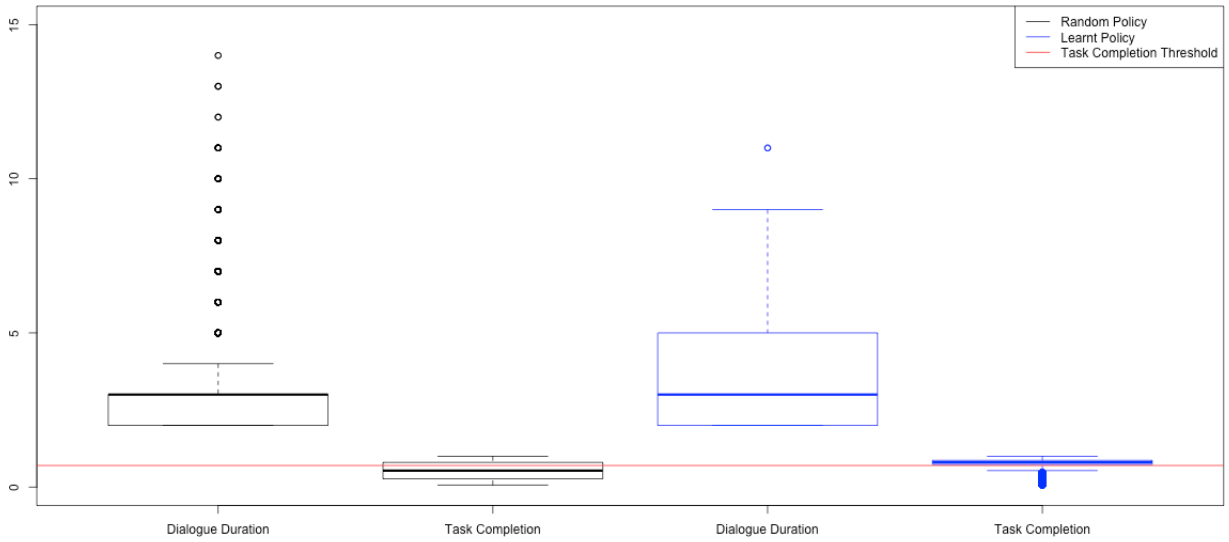


Figure 5.10: Comparison of the distributions of dialogue duration and task completion with a random policy (10000 data points) and with the policy learnt with SARSA(λ) (7000 data points).

Dialogue Duration	Task Completion	Alternative Task Completion	ASR Score	Rejections	User Satisfaction
4.87	0.75	0.94	0.66	0.34	9.55

Table 5.4: Mean Statistics on 100000 dialogues with a random system policy.

of rejections from speech recognition, task completion, and alternative task completion. The statistics for these KPI on the random corpus are displayed on Table 5.4.

In the following chapters, the simulator will be given the ability to learn from dialogues with the users. First, a state space representation will be learnt and then a reward function will be inferred.

This chapter has described NASTIA and the appointment scheduling simulator. These two systems will be used throughout the rest of this dissertation to test and illustrate the algorithms proposed for the methodology presented in Figure 1.1. Next chapter is dedicated to learning a state space representation that can handle many dialogue parameters (such as the ones listed in Table C.1) and that enables to predict efficiently the return at each state.

Part III

State Space Inference

Chapter 6

Problem Definition

As said in the introduction, the state space of an RL agent should allow to distinguish efficiently between the different returns, which are commonly defined according to user satisfaction [Dybkjaer et al., 2004, Lemon and Pietquin, 2012]. For instance, for the appointment scheduling simulator, it was important to include task completion and the number of dialogue turns in the state space because the quality of each dialogue was evaluated according to these two parameters. There has been extensive research on automatically estimating user satisfaction for a given dialogue [Walker et al., 1997a, Schmitt et al., 2011, El Asri et al., 2014a]. These studies have shown that many dialogue features (duration, mean speech recognition scores, number of help requests,...) could play an important role in user satisfaction [Larsen, 2003, Walker et al., 2002]. However, if these features were included in the state space, it would not be possible to learn by estimating the Q-function for each possible value, a parametrisation would be needed. In order to learn the user's behaviour for the appointment scheduling simulator in the previous chapter, it was chosen to use a grid-based representation. It was possible to do so because only three features were included. If more features had been included, the size of the state space would have grown exponentially to the point that learning would have become very slow. A more sample efficient representation is thus needed.

In general, in the case of adaptive SDS, it is common to define the state space according to the latest UDA and history of the dialogue (which pieces of information have been gathered, etc.). A few quantitative parameters such as the average ASR confidence score and the number of turns have also been included in the state space with representations based on radial basis functions, a neural network, or a Bayesian network [Paek and Chickering, 2005, Daubigney et al., 2012, Hatim Khouzaimi and Lefèvre, 2015]. In the more general field of RL, techniques to deal with large state spaces and continuous values have been proposed [Ratitch and Precup, 2004, Baumann and Buning, 2011, Lin and Wright, 2010, Ghavamzadeh et al., 2011]. However, these techniques have not yet been successfully transposed into the field of adaptive SDS. In this chapter, a new technique, inspired from this literature, is proposed. First, the existing methods are presented and it is explained why it was chosen not to directly apply one of them. Then, the state space representation suggested for SDS is described and applied to the mountain car problem, to the appointment scheduling simulator, and to NASTIA.

6.1 Parametrisation of the Q-function

With relatively small spaces (a few hundred state-action pairs is reasonable), a tabular representation of the Q-function can be used for learning. On the other hand, for large spaces and even more, continuous spaces, a tabular representation cannot be envisioned. In practice, in large spaces, the number of visits to one state shrinks with the number of states and in continuous spaces, the probability to encounter several times the same state is null. This problem is known as the *curse of dimensionality* [Bellman, 1957]. In the case where the Q-values cannot be stored in a table, a generalisation framework is needed. Generalisation consists of transferring the knowledge acquired in some parts of the state space to other unvisited regions. A linear parametrisation of the Q-function (Equation 6.1) is often chosen because of theoretical properties of convergence [Tsitsiklis and Van Roy, 1997] and because it is convenient to manipulate. The Q-function is expressed as a linear combination of a set of basis functions $\phi = \{\phi_1, ..., \phi_p\}$:

$$\hat{Q}_\Theta(s,a) = \sum_i^p \theta_i \phi_i(s,a). \tag{6.1}$$

This set of basis functions summarises the state space. The agent's task now is to learn the weight vector $\Theta = (\theta_1, ..., \theta_p)^T$. Note that this representation can also be used for finite MDP by defining $S \times A$ basis functions ϕ_i, each function being equal to 1 at a specific state-action pair, and 0 everywhere else.

In what follows, this representation of the Q-function will be adopted. The problem then consists of computing an approximate solution to the Bellman equation (Equation 2.7). This solution can also be seen as a fixed point of the Bellman operator T^*,

$$(T^*Q^*)(s,a) = \max_{a' \in A} \sum_{s'} P_a(s,s') \left[R(s,a,s') + \gamma Q^*(s',a') \right]. \tag{6.2}$$

A parametric approximation of the Q-function with a set of basis functions has been a focus as theoretical convergence properties can be proved [Geist and Pietquin, 2013]. Two trends can be identified. The first one assumes that the set of basis functions is known in advance. It is then possible to apply a selection scheme which aims to avoid overfitting [Ghavamzadeh et al., 2011]. The second trend does not assume that the basis functions are known but defines them by aggregating states into homogeneous clusters.

The methodology presented in this thesis supposes that the basis functions are not known in advance. Therefore, this chapter proposes a method for state aggregation. Research on learning the parametrisation of a continuous state space has first proposed dense representations of the state space. Then, policy-based and value-based methods were suggested. These 3 branches are described in the next sections.

6.2 Density-based approaches

First, *density-based* representations have been proposed [Broomhead and Lowe, 1988, Singh and Sutton, 1996, Kostiadis and Hu, 2001, Forbes, 2002, Ratitch and Precup, 2004, Mahadevan et al., 2006]. These models are built in order to span the entire state space densely, by aggregating nearby states into regions. Tile coding is an example of a density-based representation [Albus, 1971, Singh and Sutton, 1996]. States are aggregated into tiles and the state space is represented as a set of superimposed tilings with different offsets. Figure 6.1 illustrates two-dimensional tilings. Because of the offsets, each tiling discretises the state space in a unique

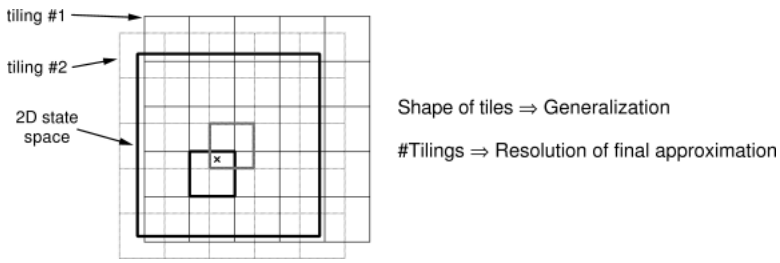


Figure 6.1: Multiple, overlapping grid tilings. Source: [Sutton and Barto, 1998]

way. Following this discretisation, the learning agent's current state, a two-coordinate vector in Figure 6.1, is determined by the tiles spanning the agent's position in this two-dimensional environment. In the previous chapter, in order to define the user behaviour of the appointment scheduling simulator, an approach similar to tile coding was chosen with only one tile. A recurrent problem with density-based representations is that they become subject to the curse of dimensionality quickly as the number of dimensions of the state space increases [Sutton and Barto, 1998]. Besides, more memory-efficient representations can be found. Indeed, it may not be useful to span the entire space because certain regions will never be visited by the learner or are not necessary to compute an optimal policy. This consideration led to two other approaches to represent large spaces, which will be referred to as *policy-based* and *value-based* representations.

6.3 Policy-based approaches

Policy-based representations intend to focus on regions of the space which are the most likely to be visited according to the agent's current policy. For instance, Bernstein and Shimkin [2008] proposed an adaptive aggregation algorithm starting with a cell covering the entire state space and splitting this cell in order to increase the resolution near the optimal trajectory. Wu and Meleis [2009] based the generalisation framework on a set of representative states which induced a Voronoi tessellation. States with the lowest visit frequencies were suppressed so that, as the agent computed an optimal trajectory, only the states in that trajectory were kept. Another paradigm, in between density-based and policy-based methods, is a model based on a Growing Neural Gas (GNG) named GNG-Q [Baumann and Buning, 2011, Baumann et al., 2012]. This work strives to compute regions of the space that are homogeneous in terms of optimal policy. In other words, states should be aggregated if they are located closely in the state space and if they share the same optimal policy. GNG-Q is density-based because the space is entirely spanned but also policy-based because of how the space is discretised.

Policy-based methods are based on the consideration that the state space should be modelled in order to represent the optimal policy of the agent. Nevertheless, under certain conditions, McCallum [1995] showed with an example that aggregating states according to their optimal policy would not necessarily be sufficient to learn an optimal policy for the task at hand. Indeed, the values of the states (or state-action pairs) should also be accounted for. The condition calling for distinguishing states according to their values is when the agent's perception of the state space is not completely accurate. For instance, in a real world application, a robot's sensor might enable the robot to perceive not all but only a subset of the environment's features. In the cases where it is unsure whether all the features necessary to learn an optimal policy are part of the agent's representation of the space, it is safer to aggregate states according to their values rather than just the optimal

action returned by the policy.

6.4 Value-based approaches

The value-based methods bypass the curse of dimensionality and aggregate states into homogeneous regions of the value function. This group contains state aggregation techniques [McCallum, 1995, Lin and Wright, 2010] and methods which approximate the value function with a parametric representation and known basis functions [Engel et al., 2005, Yu and Bertsekas, 2009, Taylor and Parr, 2009, Ghavamzadeh et al., 2011, Geist and Pietquin, 2013].

McCallum [1995] introduced Instance-Based Reinforcement Learning (IBRL) for POMDP learning. IRBL bases generalisation on a set of representative states, called instances. McCallum first proposed a density-based algorithm close to the K-Nearest Neighbours method for supervised learning. This algorithm named Nearest Sequence Memory (NSM) used a similarity measure based on history to counter the presence of perceptually aliased states[1]. The value of a newly observed state is computed as the average of the k-closest states according to this similarity metric. The value-based version of NSM used statistical tests to only keep in memory the amount of history that was necessary to distinguish between perceptions. It is not directly applicable to continuous state spaces because it requires to build a feature-based decision tree and this requires first that the feature values are discrete or have been discretised. Lin and Wright [2010] proposed a value-based tile coding representation. The tilings were arranged through genetic operations (mutation and crossover), based on the performance of the RL algorithm. Although the algorithm was shown to perform quite well on finding a memory-efficient representation, the computation time is very high for a two dimensional space, which makes it impractical for high dimensional spaces such as dialogue state spaces.

The second group of value-based methods approximates the value function with a parametric representation and known basis functions. Ghavamzadeh et al. [2011] defined the projection operator in LSTD as a Lasso problem in order to avoid over-fitting the value function. This supposes that the basis functions parametrising the value function are known. The work presented here is situated upstream, it is supposed that the basis functions Φ are unknown and it is proposed to build this set of features incrementally. Similar work proposed Gaussian Processes (GP) with temporal differences [Engel et al., 2005]. Advantages of GP is that they offer the flexibility of kernel based representations and provide an estimation of the uncertainty on the estimated Q-values. The problem with GP-based learning is that it does not scale well with the number of examples. A solution has been to a set of linearly independent vectors in the feature space, called the dictionary and then ground the computation of the mean and variance of the GP on this set.

6.5 Approaches used in the SDS literature

In the SDS literature, GP have been used for online, on-policy learning [Gašić et al., 2010, 2012, 2013]. However, it has been recently shown that, theoretically, they could also be used in a batch, off-policy setting [Chowdhary et al., 2014]. The main disadvantage of GP in the context of this thesis is that the result of learning is not easily interpretable: it would be difficult for an SDS designer to get a good feeling of what the algorithm has learnt and how the agent behaves.

[1] states that cannot be distinguished by the POMDP agent but require different agent behaviour, see [Whitehead and Ballard, 1991].

Toney et al. [2006] proposed to use an evolutionary reinforcement learning algorithm to learn a compact representation of the state space. However, no continuous feature was included in this representation which was not built for the case when rewards are partly or entirely based on user satisfaction scores.

Paek and Chickering [2005] modelled dialogue management as a sequence of decision-making modules. They performed Bayesian structure search to infer the features that were the most relevant to predict the expected discounted cumulative reward starting from each module and following the system's current dialogue management strategy. Their dialogue management model always considered the modules in the same given order. On the other hand, in our work, modules can be encountered at different times during a dialogue.

Li et al. [2009] performed basis function selection by selecting the functions which had the highest contribution to the Q-function. They needed to perform this selection for the learning algorithm that they chose to use to be efficient with a large number of dialogue features. On the other hand, the algorithm which will be proposed in the next chapter can take a large number of features as input and includes feature selection in the process of learning. Chandramohan et al. [2010b] also learnt a space representation for dialogue management. Similarly to the sparsification method employed for GP, the basis functions relied on a set of linearly independent states called a dictionary. The approach proposed in the following chapter is also based on a set of states but these states are re-arranged according to the Q-function. With this approach, the final set of states enables to interpret learning and gives valuable insights on the SDS behaviour.

6.6 Positioning

The problem to be solved is the following:

Definition 1 (State space inference problem) *Infer a value-based state space representation from a corpus of N dialogues $\mathcal{D} = (D^i)_{i \in 1..N}$ which have been manually evaluated with a numerical performance score $P^i \in \mathbb{R}$.*

The previous analysis has highlighted several questions concerning parametrised RL. First, which should be chosen between density, policy, and value-based learning? Density-based methods do not scale well with the number of features and would not be able to handle correctly the 120 features in the DINASTI corpus. Policy-based methods are not adapted to cases when the state-space is not fully observable, which is the category in which DM falls. Indeed, the correct utterance pronounced by the user is not known for sure, learning has to rely on the result of speech recognition. Therefore, it was decided to take a value-based approach.

The three main other constraints were related to the context of this thesis. Indeed, as explained before, the goal is to learn the parameters of RL so that SDS designers could benefit from it without being experts. Therefore:

1. The method has to be able to handle a large number of dialogue features of all type (discrete, continuous, categorical).

2. The result of learning should be easily interpretable.

3. The method should be usable by non machine-learning experts and therefore should not have too many parameters to set.

This chapter has reviewed the different ways to deal with a large, continuous state space in RL. Given the methodology and constraints of this thesis, some requirements have been highlighted. In order to fulfil all

these requirements, a state representation based on a Sparse Distributed Memory (SDM) [Kanerva, 1988] is proposed. The Q-function is represented as a linear combination of features as in Equation 6.1. The features ϕ_i are active on a set of representative states called *prototypes*. The set of prototypes is built and re-arranged according to the Q-function. Next chapter presents this algorithm in details and provides experimental results.

Chapter 7

Genetic Sparse Distributed Memory for Reinforcement Learning

In this chapter, a new algorithm learning a representation of the state space of an SDS is proposed. This algorithm is based on an SDM. It can handle a large amount of dialogue features and it results in a policy that can be easily interpreted by the SDS designer. This chapter starts by describing the SDM and its applications to RL. Then the combination of SDM with genetic search is presented. The solution proposed for Problem 1 is inspired from this combination. The resulting algorithm: Genetic SDM for RL (GSDMRL) is then applied to the mountain car problem, to the appointment scheduling simulator, and to DINASTI. Results on the DINASTI corpus are in the course of being published [El Asri et al., 2016a].

7.1 Sparse Distributed Memory

7.1.1 Presentation

The SDM was proposed as an alternative to Von Neumann's memory model. The SDM model was built in order to reproduce the functioning of the human long-term memory. Kanerva [1988, 1993, 2009] discovered that the distances between concepts in the human brain were like distances between points in a n-dimensional space (with n greater than a hundred dimensions). It is not necessary to keep an exact representation for a concept if it is represented as a vector in a high-dimensional space. Indeed, in high dimensions, a given point will tend to be far from most of the points so a concept will still be distinguishable from others even if the representation is not perfect. This makes for a robust model, able to handle noise.

In an SDM, each data point is represented by a binary vector of a large size (for instance, 10.000 bit vectors in [Kanerva, 1993]). The SDM links vectors of counters to binary addresses. The memory is initialised with a set of random addresses. At each address, a vector of counters is stored. Each counter corresponds to a bit in the stored data and all counters are first set to 0. The writing process is illustrated in steps 1 and 2 in Figure 7.1. Let ar be the binary address register corresponding to the data vector d to be written. The Hamming distance between ar and the addresses in the memory is computed. The addresses for which the distance is below a given threshold Δ form the *selection set* (step 1). The threshold Δ is chosen by the designer. In Figure 7.1, it is equal to 3.

Then, the data vector d is written in the vectors of counters linked to the addresses in the selection set

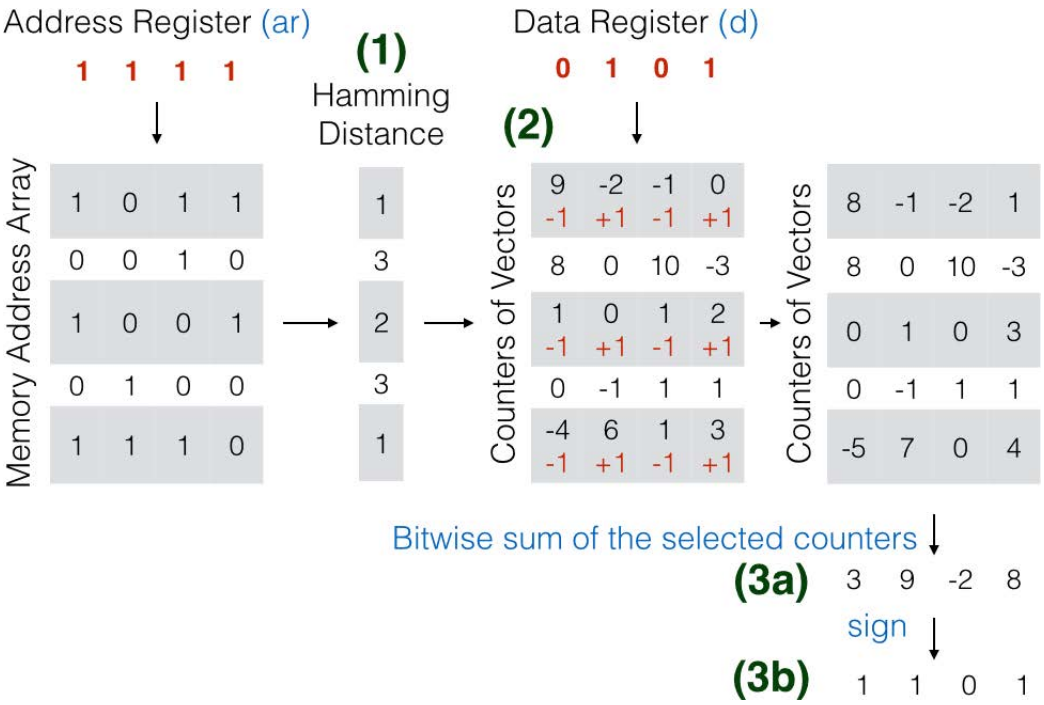


Figure 7.1: A schematic display of the reading process in a sparse distributed memory.

(step 2): each counter of each vector is incremented if the corresponding bit in d is 1 and it is decremented otherwise. Reading from the memory with ar as input is illustrated in steps 3a and 3b: the selection set is computed as before and the vectors of counters linked to these addresses are summed bitwise (3a). If a counter of the resulting sum is positive, the returned bit is 1, otherwise it is 0 (3b).

As said before, the SDM takes advantage of the fact that in large vector spaces, vectors tend to be orthogonal. When a data vector d is written with an address ar, all the addresses in ar's selection set receive a copy of d. Then, if an address ar' close to ar is presented to the memory for reading, virtually the same selection set will be computed and copies of d will be numerous in this set. Therefore, d will be output with high probability.

Figure 7.2 is a representation of Kanerva's SDM as a fully-connected feed-forward neural network. In the bottom layer, each node is a bit of the reference address (a bit equal to *1* is represented by a *1* and a *0* is represented by a *-1*). The weights between these nodes and the ones of the middle layer are the bits in the addresses of the data stored in the memory (the same representation is used). These weights are static. The Hamming distance is thus the numbers of (-1) in the vector resulting from the bitwise multiplication of the vector in the bottom layer and the one formed by the connection weights. For instance, let us give 00111 to the network, it will represent it as -1-1111. If the first node of the middle layer corresponds to, for instance 00110 (-1-111-1), the vector resulting from the bitwise multiplication will be 1111-1 and the Hamming distance will be equal to 1. This Hamming distance will then be compared to the threshold Δ and if it is lower than Δ, the middle node will be selected. The result will be the sum of the weights of all the selected middle-layer nodes which are the counter vectors. Contrary to the weights between the bottom layer and the middle layer,

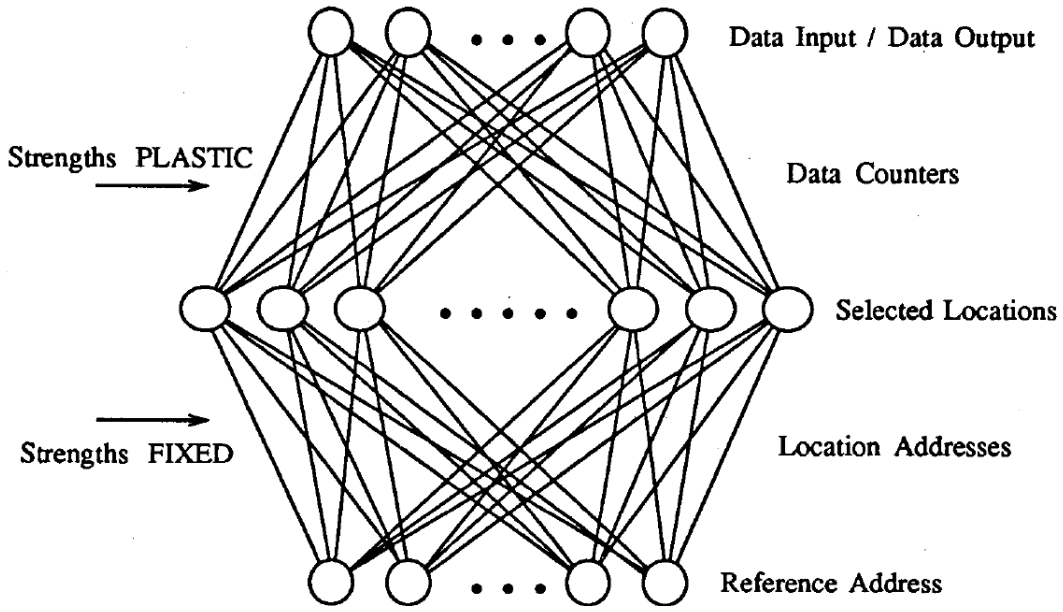


Figure 7.2: Kanerva's SDM as an artificial neural network, source: [Rogers, 1988a].

these weights are changed dynamically (each time new data is stored in the memory). RL on deep neural networks has been recently proposed [Lange and Riedmiller, 2010, Mnih et al., 2013]. The SDM approach is chosen here rather than deep networks for interpretability purposes and also because deep networks need a great amount of data for training whereas the SDM performs well on corpora of 1000 dialogues.

7.1.2 Usage in Reinforcement Learning

The SDM model has already been used to represent large RL state spaces. It is known as *Kanerva coding*, which is a term that was proposed by Sutton and Barto [1998]. The main advantage is that an SDM can handle efficiently a great number of dimensions. In this model, the addresses are state prototypes, i.e., they represent prototypical vectors of state features. The vectors of counters are no longer counters representing binary vectors but continuous values representing the Q-function.

One possible problem would be the initialisation of these addresses. Indeed, if the patterns stored in the memory are large binary vectors, a random initialisation would imply to sample a great number of addresses. For instance, in the RL context, Kostiadis and Hu [Kostiadis and Hu, 2001] built an SDM with 5000 randomly generated addresses for only 39 bit-large patterns. The problem of random initialization has been raised in other domains and methods to infer a set of addresses from data have been proposed. Hely et al. [Hely et al., 1997] introduced the SDM signal model as an alternative to the standard SDM. The SDM signal model is initialised with input data instead of random patterns. Input data are added to the memory until memory size is reached. Then, prototypes survive or are removed depending on the amount of signal they receive which in RL, would be tantamount to the state visit frequencies. In a similar vein, Anwar et al. [Anwar et al., 1999] initialised their SDM using genetic programming to infer the set of prototypes that would be the best representation of the data distribution. The fitness of the prototypes was a function of their distance to all other prototypes, measuring the

uniformity of the resulting distribution. This kind of initialisation is adapted to finding the best random set of prototypes but it is not optimised for RL as data is not likely to be uniformly distributed. Rao and Fuentes [Rao and Fuentes, 1998] applied an SDM-based space representation for robot navigation. They randomly initialised the SDM and then moved the addresses towards the observations in a similar fashion than in self-organising maps. The authors did not choose a binary representation for the data and defined the distance between two inputs as Euclidean distance. A similar approach making use of the Hamming distance was proposed by Hart and Ross [Hart and Ross, 2002]. The drawback of these approaches is that memory size needs to be specified in advance. Another method to initialise and re-arrange the prototypes for the RL problem will be proposed in our GSDMRL model.

Among the few applications of SDM to RL which have been proposed, Ratitch and Precup [Ratitch and Precup, 2004] proposed to densely cover the state space with state prototypes. They start with an empty memory. Prototypes addition is based on a heuristic: a prototype is added only if the memory is too sparse around it. If a new prototype p should be added but memory size has been reached, one or several inactive prototypes (inactive means that they are not in the vicinity of p) are removed and their two closest prototypes are replaced by an average of them. The authors chose to keep the decimal representation of the states instead of manipulating binary data. This causes high memory requirements and will require the memorisation of a discouraging number of prototypes if the state space has many dimensions. Wu and Meleis [Wu and Meleis, 2009] noticed that some prototypes were very often visited, hence too general, whereas others were too precise. A prototype being often visited means that it is too general since it is selected for many observations whereas a prototype being scarcely visited is too precise since it is only selected for a handful of observations. To deal with this phenomenon, the authors proposed to suppress prototypes that are too precise and to duplicate a prototype that is too general into a new slightly translated prototype in order to split future visits between the two. However, basing prototypes re-engineering on visit frequencies can be problematic as some states might not be visited often but still be crucial like catastrophic states or even goal states [Ratitch and Precup, 2004]. In Ratitch and Precup [2004]'s (resp. Wu and Meleis [2009]'s) work, prototype deletion (resp. addition) is made through instance averaging. When a prototype p is to be deleted (resp. added), the closest prototypes are selected and averaged so that p is replaced (resp. defined) by this average. In a way, this consists of appealing to the genetic process known as crossover. Another way to do prototype deletion or addition is by using a fitness function instead of selecting the prototypes closest to p. This alternative will be defended against Ratitch and Precup and Wu and Meleis's methods as it is based on utilities instead of density.

In order to respond to the difficulties risen by the SDM, namely the number of state prototypes and their spreading throughout the state space, the GSDMRL model is proposed. This model is inspired by Rogers' work [Rogers, 1989, 1990] on combining GSDM and genetic search for classification.

7.2 The Genetic Sparse Distributed Memory

Rogers [1989] used the sparse distributed memory model for statistical inference, combining SDM with Holland's genetic algorithms. Then, Rogers [1990] applied this *genetic memory* to the problem of weather prediction. As seen in the previous section, the weights between the bottom layer nodes and the middle layer nodes in the neural network representation of an SDM are static. The idea behind the genetic memory is that these weights, i.e., the allocation of the addresses in the location space, can be optimised according to their relevance for accurate prediction. For the weather prediction problem, the address vectors were formed concatenating feature vectors (month, pressure, cloud cover, etc.) and these addresses were then adapted through the genetic

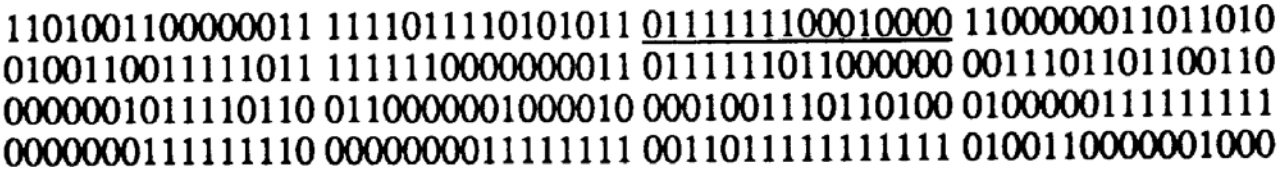


Figure 7.3: A highly-rated vector for rain prediction, source: [Rogers, 1990].

operation known as crossover. By the end of training, the bits relevant for predicting rain had been identified.

The idea of dynamically modifying the address array in order to only keep relevant addresses for prediction can be valuable for reinforcement learning. To better grasp the benefits of the genetic memory, let us reproduce here the application to rain prediction that can be found in [Rogers, 1990]. Rogers exploited samples indicating whether or not it was raining collected every four hour during 25 years on the Australian coast. He defined fifteen features (month, pressure,...) and represented each address in the genetic memory as the concatenation of these features expressed as 16-bit vectors. The data counter memorised at each address only contained one bit, equal to 0 if rain had been observed during the four hours succeeding this state and 1 otherwise. The fitness function was adapted from Rogers [1989] and measured the statistical predictiveness of each address, i.e., its relevance for rain predicting. The memory was provided with the samples in an iterative way (samples were given in groups of ten) and the crossover process was applied to automatically adjust the addresses.

After every sample had been processed, Rogers could identify the most relevant features to predict rain. To do so, he selected the addresses most highly rated by the fitness function, for example, the one given in Figure 7.3 (vector v_1). Vector v_1 is one of the most relevant vectors for rain prediction. The underlined feature is the month feature. Rogers inferred from the value of the month feature the important values for this area of the weather space. The left part of Figure 7.4 displays the Hamming distance between v_1 and the possible values for the month feature (shown on the right part of the figure). In accordance with the SDM methodology, if a new vector is addressed to the memory, the selected addresses will be the ones closest to that vector in Hamming distance. The smaller the Hamming distance between the month features, the more likely the corresponding sample vectors will be close in Hamming distance. Following this, the graph shows that the most desirable months for rain are, as expected, January, February, and March (Hamming distance of 2), and the least desirable ones are July and August (Hamming distance of 14). Besides, since the span of Hamming distances is of 12, Rogers deduces that the month feature is important since the sensibility of the Hamming distance to this field is high. It is thus possible to perform a comparative analysis of the sensibilities to the features to put forward the most important ones and exclude the least significant ones.

The use of a fitness function to select the to-be-crossovered prototypes is more coherent with the RL framework[1] than sample density-based crossover. Indeed, the goal is to build an optimal state space in the sense that it allows to predict accurately the returns. Besides, the genetic memory solves the feature selection and weighting problems as relevant features and their relevant values are put forward by the crossover process. GSDMRL adapts this work to the RL setting and proposes a new way to initially build the set of prototype states.

[1] the fitness function in the case of RL would be expressed in terms of value function prediction

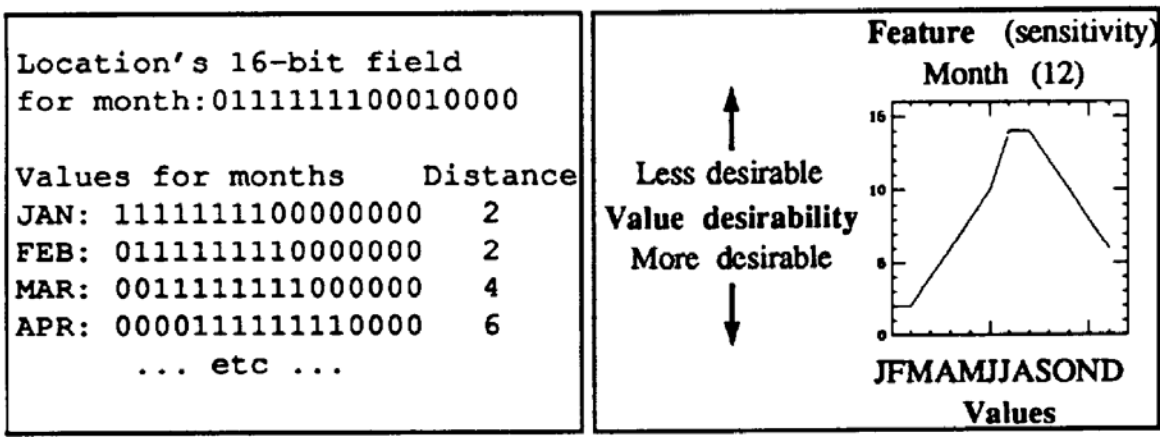


Figure 7.4: Analysis of the month feature values, source: [Rogers, 1990].

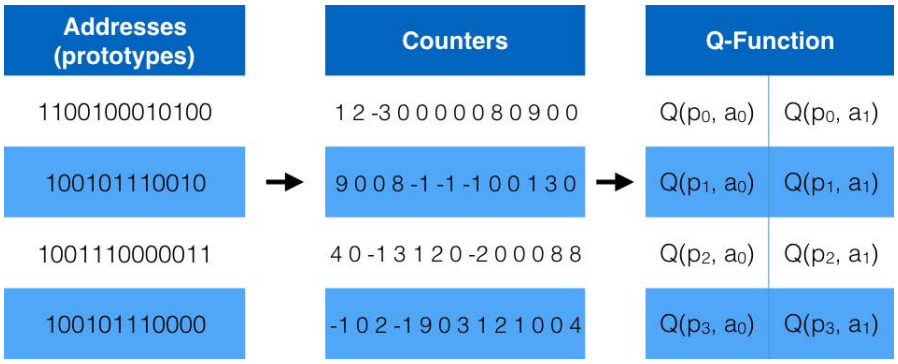


Figure 7.5: The three layers of the SDM built by GSDMRL.

7.3 Genetic Sparse Distributed Memory for Reinforcement Learning

7.3.1 Notations

A prototype p^j is composed of n bits $p^j = (p^j_1, ..., p^j_n)$. The Hamming distance between two prototypes p^1 and p^2 is $d(p^1, p^2) = \sum_{k=1}^{n}(1 - \delta_{p^1_k, p^2_k})$ where δ is the Kronecker symbol. A prototype p^2 belongs to the activation set (resp. selection set) of a prototype p^1 (resp. a state s) if $d(p^1, p^2) \leq \Delta$ (resp. if $d(p^1, s) \leq \Delta$). The notation $|S|$ will be used for the cardinal of a set S.

7.3.2 Building the Set of Prototypes

GSDMRL builds an SDM M with three layers: each prototype p^j is linked to a vector of counters $c(p^j) = c^j$ and an estimation of the Q-function for each action. A schematic view of the memory is given in Figure 7.5 with an action space of size 2. The set of prototypes is built incrementally. GSDMRL starts with an empty

memory. Each time a new state s_t is observed during learning, if the selection set X_t for this state is empty, the state is added to X_t and to the address set of M, and linked to a vector of counters and Q-values initialised to 0:

$$\forall i \in 1, ..., |A| \; M(s_t)_i = \{\forall k \in 1, ..., n \; c(s_t)_k = 0, Q^\pi(s_t, a^i) = 0\}. \tag{7.1}$$

The state s_t is then written in the counters corresponding to the prototypes in X_t, like in step 2 in Figure 7.1:

$$\forall p^j \in X_t, k \in 1, ..., n \; c(p^j)_k = c^j_k = c^j_k + (-1)^{(s_t)_k + 1}. \tag{7.2}$$

This initialisation method permits to avoid sampling a great number of random prototypes. Indeed, a state is only added as a prototype if its distance to all the other prototypes is greater than Δ. This technique allows to only cover the portion of the state space which is present in the dataset.

7.3.3 Q-function Parametrisation

As said before, the Q-function at (s_t, a^i) is represented as a linear combination of basis functions ϕ_j:

$$Q_\theta(s_t, a^i) = \sum_{j=1}^{|X_t|} \theta_{j,i} \phi_j(s_t, a^i). \tag{7.3}$$

The $\theta_{j,i}$ will be updated by the reinforcement learning algorithm. The basis functions are defined as follows:

$$\begin{aligned} \phi_j(s_t, a^i) &= \frac{w^{j,i}}{\sum_{k=1}^{|X_t|} w^{k,i}} \\ Q_\theta(s_t, a^i) &= \sum_{j=1}^{|X_t|} \theta_{j,i} \frac{w^{j,i}}{\sum_{k=1}^{|X_t|} w^{k,i}}. \end{aligned} \tag{7.4}$$

The Q-function is thus estimated as the weighted average of the values of the weights at $(p^j) \in X_t$. This weight is denoted by $w^{j,i}$. Following the same idea as the genetic sparse distributed memory proposed by Rogers, the prototypes in X_t are weighted according to their relevance to estimate the Q-values. The weight $w^{j,i}$ is a function of the fitness $f^{j,i}$ of (p^j, a^i). Rogers [1988b] showed that the weight which should be given to (p^j, a^i) is:

$$w^{j,i} = \frac{f^{j,i}}{(1 - f^{j,i})^2}. \tag{7.5}$$

The fitness score measures the relevance of p^j with respect to the regression of the Q-function. It is computed by looking at the prototypes in the activation set of p^j and checking whether the confidence intervals for the Q-function estimated at action a^i overlap with the confidence interval for $Q^\pi(p^j, a^i)$, noted $CI(Q^\pi(p^j, a^i))$. Confidence intervals are defined as [Kaelbling, 1990]:

$$\begin{aligned} CI(Q^\pi(p^j, a^i)) &= CI^{j,i} \\ &= [Q^\pi(p^j, a^i) - \epsilon^{j,i}, Q^\pi(p^j, a^i) + \epsilon^{j,i}] \\ \epsilon^{j,i} &= t^{n^{j,i}-1}_{\alpha/2} \frac{\sigma^{j,i}}{\sqrt{n^{j,i}}}. \end{aligned} \tag{7.6}$$

In this equation, $n^{j,i}$ is the number of visits to the pair (p^j, a^i), $\sigma^{j,i}$ is the standard deviation of the returns observed after visiting (p^j, a^i) and $t_{\alpha/2}^{n^{j,i}-1}$ is Student's t function with $n^{j,i} - 1$ degrees of freedom. The parameter α sets the confidence with which observed returns will be comprised in the interval. Since the prototypes in M are added only if their distance to all the other prototypes in the memory is greater than Δ, the activation set of p^j, $AS(p^j)$ only contains p^j. Nevertheless, two distant prototypes might be activated by the same states and these prototypes should then predict the same Q-values. The vectors of counters which constitute the second layer of M are used as averages of the states which activate the prototypes in the memory. Indeed, every state which activates a prototype is written in the prototype's vector of counters according to Equation 7.2. So, if two prototypes are linked to similar vectors of counters, it means these prototypes tend to be jointly activated and they should be linked to similar Q-values. The fitness $f^{j,i}$ of (p^j, a^i) is computed by first finding the activation set of p^j's vector of counters c^j. Then, c^j is compared to the other vectors of counters in Hamming distance and the vectors closer than Δ form the activation set $AS(p^j)$. The fitness of (p^j, a^i) is the ratio of the number of confidence intervals overlapping on the size of $AS(p^j)$:

$$f^{j,i} = \frac{|\{p^k \in AS(p^j)\ ,\ (CI^{j,i} \cap CI^{k,i}) \neq \emptyset\}|}{|AS(p^j)|} \tag{7.7}$$

Besides weighting the prototypes, the fitness scores are also used to re-engineer the prototypes in M, which is explained in the next section. Figure 7.6 describes the process of computing the selection set for a state s_t and a Δ of 14 and then computing the Q-value for (s_t, a_i) as a linear combination of the values of the prototypes in the selection set.

7.3.4 Re-engineering the Prototypes

Let us define the set-policy π_s for a prototype p^j as:

$$\pi_s(p^j) \in \underset{a}{\operatorname{argmax}}\, Q^\pi(AS(p^j), a^i) \tag{7.8}$$

$$\in \underset{a}{\operatorname{argmax}}\, \frac{\sum_{k=1}^{|AS(p^j)|} w^{k,i}\theta_{k,i}}{\sum_{k=1}^{|AS(p^j)|} w^k} \tag{7.9}$$

The re-engineering rule is only based on the confidence interval for the set-policy of each prototype. The fitness score of (p^j, π_s) measures the relevance of p^j for predicting $Q^\pi(s, \pi_s(p^j))$ where s is a state activating p^j. A prototype with a low fitness adds noise to learning. Therefore, the prototype p^- with the lowest fitness score for its set-policy is suppressed and replaced by the result of a random crossover between $c^{+,1}$ and $c^{+,2}$ which are respectively the vectors of counters for the two fittest prototypes $p^{+,1}$ and $p^{+,2}$. The crossover process is detailed in Algorithm 1. The suppression of a prototype is not disastrous for the memory as only prototypes with an activation set of size greater than one are suppressed. Thus, there will remain other, fitter prototypes in their regions of the state space. A separate counter is also incremented each time a prototype is involved into a crossover operation so that if there are several prototypes with the highest fitness score, the ones that have been the less involved in crossover operations would be chosen. A particularity of GSDMRL is that during learning, some prototypes are added and some are suppressed. Right after a prototype has been added to the state space, if it is part of a selection set, it does not contribute to action selection via Equation 7.4. It only starts contributing after having been visited and its Q-values having been updated.

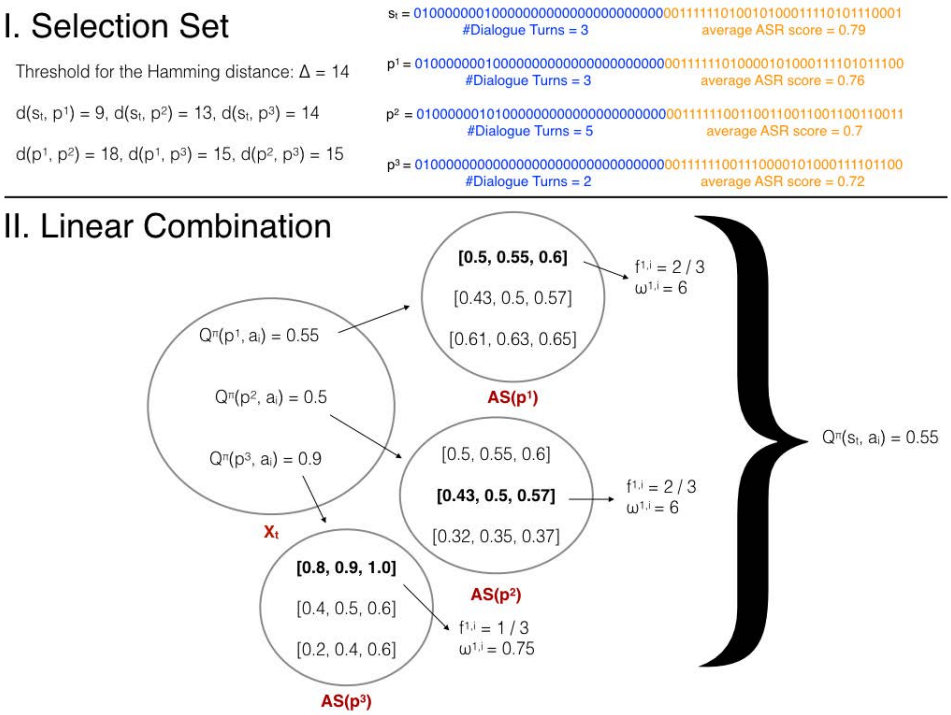


Figure 7.6: Computing the Q-value for a state action pair (s_t, a_i) with GSDMRL.

Algorithm 1 Crossover process

Choose random crossover size s in $1, ..., n$
Choose random crossover point c in $1, ..., n - s$
$p^{\text{cross}} = \text{concat}(c^{+,1}_{1,...,d}, c^{+,2}_{d+1,...,d+1+s}, c^{+,1}_{d+s+2,...,n})$
Add p^{cross} to the set of addresses in M
Remove p^{-} from M

Once the radii of the confidence intervals (ϵ in Equation 7.6) have reached a low value, the confidence intervals in an activation set will no longer overlap. Therefore, a parameter β is added to the algorithm as a lower bound for the confidence intervals. When the radius of a confidence interval goes under β, it is set to β. Algorithm 2 is a general algorithm for online RL with GSDMRL.

Algorithm 2 Online reinforcement learning with GSDMRL

Require: Hamming distance threshold Δ, lower bound for confidence intervals β

```
Start with M = ∅
while episodes are added do
    while episode not finished do
        At time t, observe s_t
        Compute the selection set X_t of s_t
        if X_t is empty then
            Add s_t to X_t, add s_t to the address set of M
        end if
        Write s_t in the vector of counters of the addresses in X_t
        Compute the Q-function at each a ∈ A as the weighted average of the Q-function at the state-action
        pairs in X_t
        Choose action a_t depending on exploration strategy
        Observe reward r_t, state s_{t+1}
        Update the Q-function
    end while
    Update confidence intervals and fitness scores
    If necessary, do crossover
end while
```

7.3.5 Setting the two parameters

There are two parameters to set for GSDMRL: the threshold for the Hamming distance Δ and the lower bound for the confidence intervals β.

In all the following experiments β is set to either 0.1 or 0.01. This parameter specifies when two Q-values should be considered close enough to have the same meaning. It depends on the scope of the rewards given to the system. For the mountain car task, a negative reward equal to -1 is given after each unsuccessful turn. Therefore, if the car takes n steps to park on top of the hill, the return r_0 will be equal to $\sum_{i=0}^{n} -\gamma^t$, i.e, $-\frac{1-\gamma^n}{1-\gamma}$. For n = 1, r_0 is -1, for n = 2, r_0 is 1.99, for n = 3, r_0 is 2.97, for n = 4, r_0 is 3.94, etc. This distribution of rewards allows to distinguish values at 0.1 and does not require finer precision. On the other hand, the rewards given to the appointment scheduling simulator and NASTIA require finer precision because they are of the type $\gamma^t P$ with P the performance score. Therefore, β is set to 0.01 for these applications. For the methodology proposed in this work and according to the reward functions suggested in the following part, it is advised to set β to 0.01.

The parameter Δ determines the number of prototypes wthat will be created. In general, it is good practice to set the threshold a little above the average distance between the points in the space [Kanerva, 1993]. The reason for this relies on probabilities in large binary spaces. There are two possible settings: online and batch.

In the online case, illustrated here with mountain car, it is possible to tune Δ in order to find the minimal set of states ensuring optimal performance. In the batch dialogue setting, it is not possible to test for performance on new dialogues. However, it is possible to divide the corpus of rated dialogues into a training and a test set. The training set serves to train GSDMRL and then the test set is used to evaluate the expected return with the learnt state space. This approach is illustrated in Section 7.4.3 to compare performance between GSDMRL and a grid-based representation. For this parameter, it is thus advised to start by setting it around the average distance between the points and then follow the approach in Section 7.4.3 to find the smallest set of prototypes for the highest possible performance.

In the next section, GSDMRL is applied to the mountain car task with SARSA(λ) as learning algorithm and then it is applied on the corpus of dialogues collected with the appointment scheduling simulator and DINASTI with Fitted-Q Iteration (FQI, [Gordon, 1995]). FQI is a batch RL algorithm. Compared to online learning, batch RL makes a more efficient use of the available data and entails more stable solutions [Lange et al., 2012]. FQI approximates the optimal Q-function from a fixed set of data samples. This algorithm does not depend on any tunable parameter and it exploits efficiently the generalisation capability of any supervised learning algorithm. It was shown to perform well on dialogue management [Chandramohan et al., 2010a]. The optimal Q-function Q^* is the solution of the Bellman equation (Equation 2.5) and can also be seen as fixed point of the Bellman operator $T*$:

$$Q^* = T^*Q^* \tag{7.10}$$

T^* admits a unique fixed point [Bellman, 1957] which can be incrementally computed via *Value Iteration*: $\hat{Q}_i^* = T^*\hat{Q}_{i-1}^*$. FQI learns from a set of samples $\mathcal{D} = (s_i, a_i, r_i, s_i')_{1 \leq i \leq N}$. The weight vector θ in the linear parametrisation of the Q-function (see Equation 6.1) is found by minimising the mean square error in the following equation:

$$\text{MSE} = \frac{1}{|\mathcal{D}|} \sum_{(s_i,a_i,r_i,s_i') \in \mathcal{D}} (y_i - \hat{Q}(s_i, a_i))^2 \tag{7.11}$$

$$y_i = r_i + \max_a \hat{Q}(s_i', a) \tag{7.12}$$

An iterative solution to this problem is:

$$\begin{aligned} \theta_j &= (\sum_i^N \phi_i \phi_i^T)^{-1} \sum_i^N \phi_i y_i^j \\ y_i^j &= r_i + \gamma max_a \theta_{j-1}^T \phi(s_i', a) \end{aligned} \tag{7.13}$$

7.4 Experiments

This section has two main goals:

1. Validate GSDMRL on the mountain car problem and on the appointment scheduling simulator by comparing it to tile coding. Tile coding was chosen for comparison because it has been proven to be powerful for the mountain car task [Timmer and Riedmiller, 2007]. Moreover, as GSDMRL, tile coding also only takes two input parameters, namely the number of grids (tilings) and the number of cells (tiles) per grid, which makes it relatively easy to set and comparable to GSDMRL.

2. Illustrate the scalability and interpretability of GSDMRL on the DINASTI corpus (120 dialogue features).

7.4.1 The Mountain Car Task

In this section, GSDMRL is illustrated in an online (SARSA(λ)) setting. The prototypes in the memory are the concatenation of the binary representations of the position and velocity of the car. Both are 32-bit vectors, resulting in 64-bit large addresses. The threshold for the Hamming distance was set to 29, a little above the average distance between the points in the space which is 26. The lower bound for the 95% confidence intervals β was set to 0.1. The discount factor γ was equal to 0.95. In SARSA(λ), λ, the discount rate for the eligibility traces was set to 0.95 and α, the learning factor, was equal to 0.05. The car always started in the valley, with a position of -0.5 and a null velocity. These conditions make the task harder than a random initialisation of the position as the exploration of the state space must be performed by the agent. 100 trials of 500 episodes were run. After each episode, the agent's current policy was tested on one episode. Learning is measured in terms of the average, on the 100 trials, of the number of steps needed to reach the goal.

Results and Discussion

In Figure 7.7 are displayed the learning results of GSDMRL and tile coding with SARSA(0.95) and an ϵ-greedy exploration, with ϵ equal to 0.001. Following this strategy, with probability ϵ, a random action is performed and with probability $1 - \epsilon$, the car follows its current optimal policy. Tile coding was tested with 2 tilings of 4×4 tiles. The offsets of the tilings were randomly drawn for each trial. With Δ equal to 29, an average of 15 prototypes were created by GSDMRL. The smallest memory had 12 prototypes and the largest one had 21 prototypes. To compare similar memory sizes, tile coding with 2 tiles of 3×3 tilings was tested on the same task. With such low resolution, a good policy could not be found. When the tiles become too large, the state space loses its Markovian property as one or several steps might be needed to transit from one tile to another.

To give a better idea of the scaling properties of tile coding and GSDMRL, the same problem was tested with actions of -1N, 0N and 1N. This task is harder than the previous one as it takes more steps to reach the

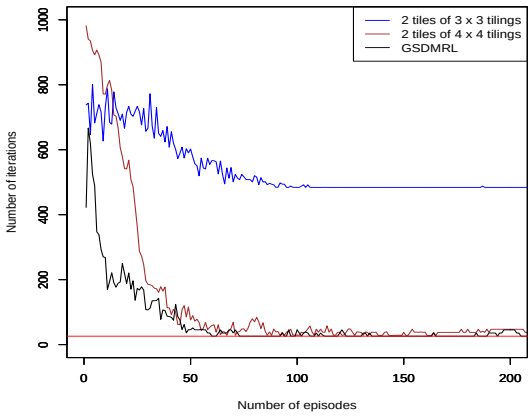


Figure 7.7: Learning results of SARSA(0.95) with tile coding and GSDM.

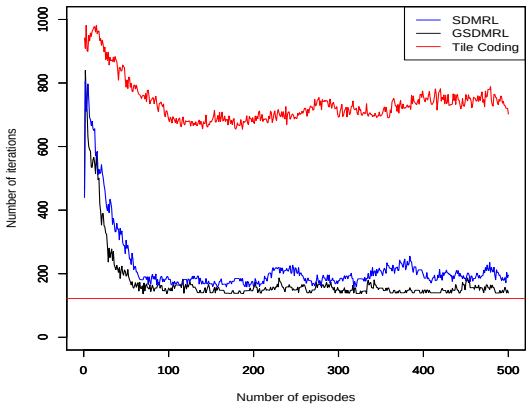


Figure 7.8: Learning results of SARSA(0.95) with tile coding, GSDM and SDM with less powerful actions.

goal so exploration takes more time. The results are displayed on Figure 7.8. In average 20 prototypes were created by GSDMRL with a maximum of 36 prototypes. GSDMRL converges to the optimal policy in less than 100 runs whereas tile coding with 32 tiles almost never found a policy that achieved the desired goal. The third curve on this Figure corresponds to GSDMRL without the crossover operations (SDMRL). Learning time is slower and the policy learnt is of a worse quality than the one learnt with GSDMRL.

Figure 7.9 displays the optimal policy at the states stored in the observation memory. The interpretability of the policy learnt with GSDMRL is here straightforward because the state space has only two dimensions. Three clusters can be easily distinguished, which explain the agent's behaviour. When the agent starts with a negative position and velocity, it has to go left, up the opposite hill then, once the velocity is positive, the car has to accelerate and once both the position and velocity are positive, the car should not apply any force to reach the top of the hill with a quasi-null speed. An optimal policy was found with only three prototypes, distributed in these three clusters. The posterior analysis of the optimal policy can thus help tune the threshold Δ to reduce memory requirements to a minimum.

7.4.2 Application to the Appointment Scheduling Simulator

In this section, GSDMRL is applied to dialogue management via the appointment scheduling simulator. GSDMRL is applied in a batch setting, with FQI as learning algorithm. The methodology is as follows:

1. The set of prototypes is initialised on the corpus of dialogues collected with a random system policy.
2. GSMRL and FQI are applied on this corpus in order to re-engineer the prototypes.
3. The set of prototypes and their weights are then used to learn a policy on new dialogues generated with the simulator. For this online learning phase, SARSA(λ) is used with an ϵ-greedy policy where ϵ is set to 0.01 and λ is set to 0.9. If, for a given state observed during learning, the selection set is empty, it was chosen to add to the selection set the closest prototype to the state in Hamming distance.

In order to learn a tile coding representation:

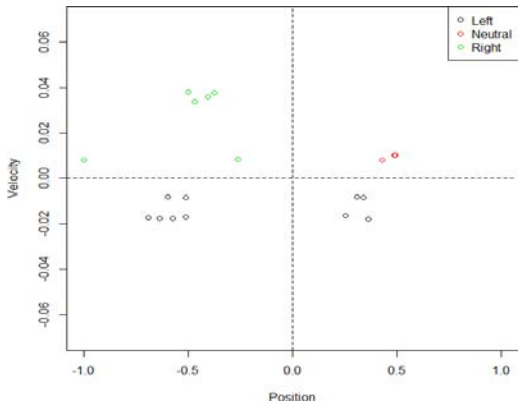


Figure 7.9: Optimal policy learnt with GSDM.

1. Entropy-based discretisation is applied on the corpus of dialogues collected with a random system policy.
2. This discretisation defines a grid over the dialogue features.
3. The grid is then used to learn a policy on new dialogues generated with the simulator. Similarly, for this online learning phase, SARSA(λ) is used with an ϵ-greedy policy where ϵ is set to 0.01 and λ is set to 0.9.

For GSDMRL, a dialogue state was the concatenation of 32-bit vectors, each vector being the binary representation of one of the 5 features: dialogue duration, system task completion, alternative system task completion, mean ASR confidence score, and number of ASR rejections. The vectors were thus 160-bit large. The lower bound for the 95% confidence intervals β was set to 0.01. The threshold Δ was set to 35. The results of learning are compared according to several parameters among which the system's score system task completion + user satisfaction. It is recalled that system task completion corresponds to the preference score of the system for the slot accepted by both participants. User satisfaction is the SVOR score depending on: the number of dialogue turns, the number of ASR rejections, the mean ASR confidence score, and the user task completion. The other dialogue parameters are statistics on the features used to define the state space, namely: the number of dialogue turns, the number of ASR rejections, the mean ASR confidence score, the system task completion, and the alternative system task completion (the highest preference score the system could obtain). Finally, the number of states created by both methods and the learning times are also compared.

The methodologies presented for GSDMRL and tile coding are tested on 50 runs for different sizes of the corpus of dialogues collected with a random system policy: 500, 1000 and 2000. Table 7.1 presents the statistics for the dialogue parameters and Figure 7.10 displays the learning results for GSDMRL and tile coding.

Results and Discussion

The difference between this setting and the previous one is that in the previous setting, the representations of the state space were built during learning whereas in this setting, the representations are built on the corpus of dialogues with a random policy and then a policy is learnt and evaluated based on these representations.

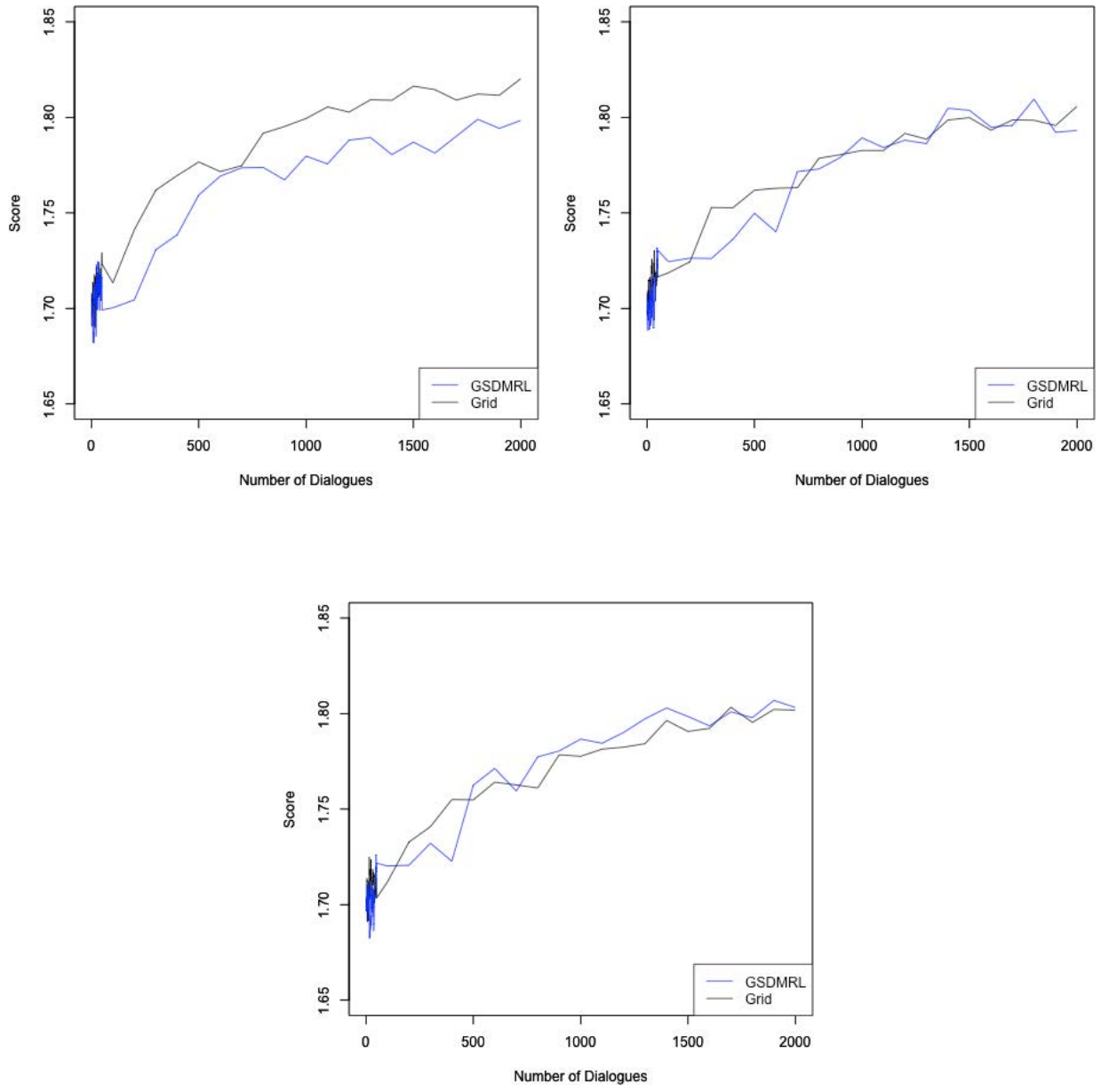


Figure 7.10: Learning results of GSDMRL and a grid-based representation after learning, from top left to bottom left, on corpora with 500, 1000 and 2000 dialogues (50 runs).

Model	Dialogue Duration	Task Completion	Alternative Task Completion	ASR Score	Rejections	User Satisfaction	Number of States
500 Dialogues							
Grid	6.78	0.89	0.92	0.66	0	9.4	152.9
GSDMRL	6.98	0.88	0.91	0.67	0.36	9.31	11
1000 Dialogues							
Grid	6.75	0.87	0.92	0.66	0	9.42	251.5
GSDMRL	7.0	0.87	0.91	0.66	0.31	9.34	14.2
2000 Dialogues							
Grid	6.56	0.86	0.93	0.66	0	9.45	293.3
GSDMRL	7.17	0.88	0.89	0.67	0 .332	9.30	16.7

Table 7.1: Mean Statistics on 100 dialogues after learning on corpora with 500, 1000 and 2000 dialogues (50 runs).

As shown in Figure 7.10 for training sets superior to 500, GSDMRL learns a policy which performs similarly to FQI on the grid. Table 7.1 shows that this result is achieved with significantly less states than with the grid representation. Indeed, the number of states goes from 153 to 293 with the grid representation whereas it goes from 11 to 17 with GSDMRL. These results show that GSDMRL scales well with the number of dialogues. Table 7.1 also shows that GSDMRL performs a different tradeoff between system task completion (0.88 for GSDMRL vs. 0.86 for the grid) and user satisfaction (9.30 for GSDMRL vs. 9.45 for the grid). The difference in user satisfaction scores is probably due to a higher use of ASR rejections by GSDMRL compared to the grid. Theses results show that it is possible to use GSDMRL in an online context, without any memory update and achieve similar results than the grid with about 17 times fewer states. In the next section, GSDMRL is compared to grid-based FQI on the DINASTI corpus.

7.4.3 Application to DINASTI

This section presents the results of two experiments. The first one compares GSDMRL to entropy-based discretisation on a small set of features. In both cases, the learning algorithm was FQI. For GSDMRL, a dialogue state was the concatenation of 32-bit vectors, each vector being the binary representation of one of the 120 features in the DINASTI corpus. The lower bound for the 95% confidence intervals β was set to 0.01. The discount factor γ was equal to 0.99. The grid-based representation cannot be used on all the 120 features because the size of the state space, defined as a Cartesian product of the intervals, would become inconveniently high. Therefore, to compare these two approaches on DINASTI, the first experiment only used two features: number of dialogue turns and the mean speech recognition confidence score. GSDMRL was thus composed of 64-bit vectors and Δ was set to 15.

The second experiment highlights the scalable and interpretable powers of GSDMRL. To do this, the original 120 features of the DINASTI corpus were included in the state space and the threshold Δ was set to 150.

Training Corpus	Test Corpus	#states Grid	#states GSDMRL	$\hat{r}_0^{\pi_{Grid}}$	$\hat{r}_0^{\pi_{GSDMRL}}$
531	1200	427.07	**63.9**	7.867	7.878
731	1000	282.23	**66.67**	7.868	**7.885**
831	900	271.99	**66.85**	7.869	**7.891**
931	800	266.84	**62.58**	7.872	**7.900**

Table 7.2: Comparison of GSDMRL and a grid-based representation on the DINASTI corpus. All values are averages on 500 runs. The results in bold are statistically significant (p-value under 0.01 for Student's t-test).

Experiment 1: Comparing GSDMRL to a Grid-Based Representation

The comparison of GSDMRL and the grid is based on the expected cumulative reward for each dialogue in DINASTI. Each dialogue started with deciding the strategy negotiation so each dialogue started with the same state s_0. This state s_0 was the first prototype to be written in GSDMRL and it was added to the grid. For both representations, it was ensured that this state would only be visited (or selected in the case of GSDMRL) at the beginning of each dialogue. The reward function gave $\frac{P_i}{\gamma^{t_f-1}}$ at the end of each dialogue D_i where P_i is the performance rating for D_i and t_f is the last dialogue turn[2]. For the other intermediate dialogue turns, the reward was equal to 0. These rewards were designed so that the value of the Q-function at the initial state would be an estimation of the expected performance for the dialogue:

$$\begin{aligned} Q^{\pi}(s_0, a) &= E[r_t = \gamma^{tf-1} \times \frac{P_i}{\gamma^{t_f-1}} \mid \mathcal{D}, s_0, a, \pi] \\ Q^{\pi}(s_0, a) &= E[P_i \mid \mathcal{D}, s_0, a, \pi] \end{aligned} \tag{7.14}$$

The value which serves to compare the two representations is $\hat{r}_0$, the expected cumulative reward starting from state s_0 and following the distribution of actions in the corpus, that is to say:

$$\hat{r}_0^{\pi} = \sum_a \frac{n(s_0, a)}{\sum_a n(s_0, a)} Q^{\pi}(s_0, a), \tag{7.15}$$

where $n(s_0, a)$ is the number of visits to (s_0, a).

In order to compare GSDMRL and the grid on an equal basis, the policies learnt with both representations were evaluated on the same state space representation. Two policies π_{GSDMRL} and π_{Grid} were learnt with GSDMRL and the grid representation respectively. Then the expected cumulative reward for both policies was computed by projecting the policies on the simple state space representation composed of NASTIA's dialogue phases. Thus, $\hat{r}_0^{\pi}$ was computed as the expected cumulative reward given the policy π_{GSDMRL} or π_{Grid} and given the probability transitions between the dialogue phases. This evaluation tests which mapping from dialogue phases to a higher-dimensional space is the most efficient.

Experiment 2 : Scalability and Interpretability of GSDMRL

In this second experiment, a policy is learnt with 120 features. It is possible to easily analyse the policy, based on the prototypes. In order to achieve this, for each action a, all the prototypes whose optimal action is a are

[2]Dialogue turns start at time $t_0 = 0$

grouped together. Then, the mean feature-wise Hamming distance is computed between the prototypes of one group and the prototypes of the other groups. These distances are normalised by the variances of the values in the corpus. The highest distances are representative of the features which play an important role in choosing action a. For instance, for the negotiation strategy, if the mean speech recognition score of the prototypes choosing the listing action is, in normalised distance, far from the mean recognition scores of the prototypes choosing another action, then it means that the mean speech recognition score is important when it comes to choosing listing. Prototypes with a certain range of recognition scores will tend to have listing as optimal action. Following this consideration, the policy can be analysed by identifying the important features and how they influence the system's behaviour.

Experiment 1: Results

The results of this experiment are presented in Table 7.2. The first observation is that the number of states varies considerably less with GSDMRL than with the grid. The number of states in GSDMRL is also significantly lower with this value of Δ. Even with a significantly smaller number of states, the average expected cumulative reward with GSDMRL is at least as good as the one with the grid and even a little higher. This shows that GSDMRL builds a small set of states upon which a good policy is learnt in a more efficient way than it is possible to do with a grid-based representation.

Experiment 2: Results

A total number of 899 prototypes were added to the memory. An analysis of these prototypes highlights the most important features in the policy NASTIA learnt with GSDMRL. These features and their values explain the circumstances behind the system's decisions.

Let us consider the phase where the negotiation strategy is chosen. Let us call $M(LA), M(SI), M(UI)$ the vectors of counters which are linked to the respective optimal actions listing, system initiative, and user initiative. An analysis of the difference between the values of $M(LA)$ and the ones of $M(SI)$ and $M(UI)$ enables identification of the most relevant features for decision making. The mean difference was computed for each feature and then normalised by the variance of the values in the corpus. An example is given in Figure 7.11.

Concerning the negotiation strategy phase, the listing action is linked to dialogues that have not lasted long (35.23 seconds *vs.* 67.45), where a few rejections from speech recognition have been observed (number of rejections equal to 1.26 *vs.* 4.68), and where, in average, less than one confirmation has been asked to the user. This reflects the fact that this strategy should rather be chosen at the beginning of a dialogue. As expected, the system initiative strategy is the one that should be chosen if the dialogue is problematic. In average, system initiative should be chosen after 5.43 speech recognition rejections *vs.* 3.92 for user initiative and 1.26 for listing. User initiative seems more fit after a list of availabilities was proposed with listing but none of the propositions suited the user. As for the phase recovering from ASR rejections and user inactivities, the two most important features are the number of user dialogue turns and dialogue duration. It is better to ask the user to repeat after dialogue length has gone above a certain threshold (dialogue duration equal to 183.42 seconds *vs.* 52.97). The implicit confirmation strategy is chosen after dialogue duration has gone above 200 seconds and when, in average, there has been less than 1 rejection from speech recognition. The explicit strategy should be chosen if more than 2.5 help messages have been played and the number of user turns is high. This means that the dialogue might be problematic and it is better to choose a more conservative strategy. Informations

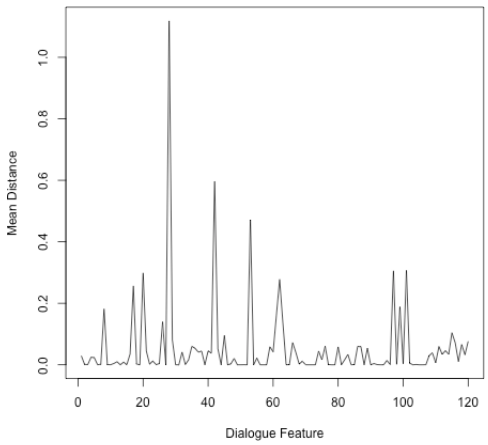


Figure 7.11: Mean normalised distance between the vectors of counters for which the optimal action is listing and the vectors of counters for which the optimal action is either user initiative or system initiative.

about the system's calendar should not be provided to the user if the system has already tried the listing strategy at least once. This reflects the fact that the user is already partially aware of system availabilities and might be annoyed by a reminder. On the other hand, informations are given after, in average, 0.47 dialogue acts notifying the user that a slot is not available have been observed. This means that after user initiative or system initiative, if the user proposes unrealisable constraints, the system should always provide informations about its calendar. Finally, indications of a long dialogue with several unsuccessful negotiation rounds should push to choose the shortest help message.

This chapter has introduced a novel value-based state aggregation technique , GSDMRL, combining a sparse distributed memory with genetic search. GSDMRL was shown to perform better than a grid-based representation on three different tasks. GSDMRL produced smaller state spaces which led to better or equivalent performance as higher-resolution grids. It was also shown that GSDMRL could be applied to high-dimensional spaces and that the result of learning with GSDMRL was easy to visualise and interpret. The next chapter proposes algorithms which learn a reward function from an annotated corpus and given a state space representation. These algorithms are tested and compared on LTI and the appointment scheduling simulator.

Part IV

Reward Function Inference

Chapter 8

Problem Definition

This chapter presents two algorithms which learn a reward function from the corpus of scored and annotated dialogues. The first chapter states the problem and describes the state of the art. The second chapter presents one solution to learn a reward function from a set of rated dialogues. It also compares several estimators of subjective dialogue performance because such estimator is needed for the algorithm. The third chapter presents another solution which does not need an estimator and compares the two reward functions on LTI.

8.1 The Problem

The reward function inference problem is posited as such:

Definition 2 (Reward function inference problem) *Infer a reward function from a corpus of N dialogues* $\mathcal{D} = (D^i)_{i \in 1..N}$ *which have been manually evaluated with a numerical performance score* $P^i \in \mathbb{R}$.

This definition leaves room to several interpretations. The characterisation of a reward function solution to this problem necessitates an interpretation of the performance scores and of the role the scores should play in learning. Two views have been exposed in the literature and are presented in the next section.

8.2 Performance Score Interpretation

8.2.1 Utility of the final dialogue state

The first trend construes the performance scores as the utility of the final dialogue state, that is to say, the reward function should give null rewards throughout the dialogue and give the performance score as final reward, at the end of the dialogue [Walker et al., 1998, Rieser and Lemon, 2011]. This reward function is the following:

$$R : (s, a, s') \in S \times A \times S \mapsto \begin{cases} 0 & \text{if s' is not the final state of the dialogue} \\ \hat{US} & \text{otherwise} \end{cases} \tag{8.1}$$

Under this interpretation, the scores relate to the return in the sense that the objective function at time t denoted r_t is equal to the performance score discounted relatively to the number of dialogue turns which have occurred between t and t_f, where t_f is the final dialogue turn: $r_t = \gamma^{t_f - t} P$. This objective function impels to learn a

policy which minimises the time spent on the task and maximises the performance. In this configuration, if the discount factor γ is set to 1, the objective function is equal to the performance score at each time step.

Walker et al. [1998] applied RL with the function in Equation 8.1 to a corpus of dialogues with ELVIS (EmaiL Voice Interactive System), a phone-based system which can read Emails and summarize the content of an inbox (by sender and subject). At the beginning of each dialogue, ELVIS has to choose between System Initiative (SI) and Mixed Initiative (MI). When SI is chosen, the system will guide the dialogue until the end and the user will be asked to specify each parameter (sender, subject,...) once at a time. MI, on the other hand, leaves the user free to specify as many arguments as wanted in her utterances. Other actions the system has to decide concern the presentation of the Emails to the user (e.g., read the messages or summarize the content of the inbox). Rieser and Lemon [2011] applied the function in Equation 8.1 to an in-car system which enabled the user to search through a musical library and play songs. They evaluated the RL-based strategy with real users and showed that this strategy was rated significantly higher than a baseline.

8.2.2 Ranking

A second possible interpretation is to infer a ranking from the scores and learn a function which emulates this ranking [Sugiyama et al., 2012]. This approach was motivated by the case where several annotators would determine the dialogues' performance scores. In this context, rankings are preferred to absolute performance scores because they are more consistent. The authors' argument is that rankings bypass the difficulties linked to the different possible interpretations of the Likert scale. The performance scores are used to compute *preferences*: pairs of dialogues are ranked according to the difference between their two performance scores. A reward function is then learnt in order to reproduce the same ranking between the dialogues. To do this, a performance score is interpreted as the sum of the Q-values of the state-action pairs visited during the dialogue. At each step n of the algorithm, under the current reward function θ_n, the estimated performance score for a dialogue i is $e_i^{\theta_n} = \sum_t Q^{\theta}(s_{i,t}, a_{i,t})$.

The ranking learnt with the reward function returned by the algorithm was compared to the rankings given by two experts, on the basis of agreement rate and correlation. The agreement rate and correlation were close to the ones between the two annotators. The agreement rate (resp. correlation) between the annotators was equal to 0.632 (resp. 0.211) whereas it was equal to 0.600 (resp. 0.200) between the annotators and the inferred ranking.

8.2.3 Model Evaluation

A natural interrogation arising from these different ways of treating performance scores is: on what basis should these two models be compared? The reward inference problem as stated in Definition 2 is ill-posed in the sense that the solution to this problem is not unique since different reward functions can be inferred from performance scores. Nevertheless, a natural metric to compare different reward functions is the average performance of the policies learnt with these reward functions. The higher the performance under a given policy, the better the corresponding reward function. The two previously mentioned methods do not include policy performance in the process of learning the reward function, they infer a function which reflects the scores given a certain distance metric (Euclidean distance for [Walker et al., 1997a] and Spearman rank correlation coefficient for [Sugiyama et al., 2012]). A principled manner to integrate policy performance in the learning algorithm is by performing Inverse Reinforcement Learning (IRL). The IRL problem was defined as follows [Russell, 1998, Ng and Russell, 2000]:

Given 1) measurements of an agent's behaviour over time, in a variety of circumstances; 2) if needed, measurements of the sensory inputs to that agent; 3) if available, a model of the environment.
Determine the reward function being optimised.
In other words, the reward function learnt with IRL describes the task followed by an expert agent. An optimal policy learnt with this reward function would be equivalent to the policy followed by the expert.

Paek and Pieraccini [2008] first suggested to use IRL on human-human dialogues in order to learn a reward function that would enable the SDS to mimic the behaviour of human operators. In the same vein, Boularias et al. [2010b] learnt a reward function for an RL-based SDS from human-human dialogues, in a Wizard-of-Oz (WOZ) setting. However, WOZ-emulated behaviour is only preferable to statistical learning when it is clear that the human can always make the best choice. It is not chosen to pursue this line here, given the constraints of this thesis.

Before presenting the design choice made in this work, it is necessary to discuss the usage of the discount factor γ with the two interpretations presented in the previous sections.

8.2.4 Of the usage of a discount factor

The discount factor was first introduced in the field of RL in order to avoid infinite returns. However, dialogue management is an episodic RL task. Therefore, it would be possible to set the discount factor γ to 1. Yet, it is common to set γ between 0.9 and 0.99. There can be two reasons for this choice. The first one is that rewards are distributed to the system throughout the dialogue, not necessarily only at the end. For instance, within-dialogue negative rewards can be distributed for ASR rejections or user time outs [Laroche et al., 2011]. In this case, setting γ below 1 allows to give more importance to immediate rewards, which are direct feedback for the latest action. The other possible reason is when only a final reward is given to the system. For instance, let us consider that an estimation of the performance score is given at the end of the dialogue. Then, if γ is set to 1, the optimal policy will optimise the expectation of the performance score, at each state. By adding a discount factor, the return is discounted according to the time on task, which, as said earlier, jointly maximises the performance score and minimises dialogue duration. This other possibility should however be considered cautiously. Indeed, minimising dialogue duration only makes sense if user expertise is accounted for in the state space. In effect, it is likely that a dialogue with a novice user will be longer than one with an expert user. In this case, dialogue length is not necessarily representative of dialogue management quality. However, if an estimator of user expertise is integrated into the state space, then minimising dialogue duration will result in minimising duration for each type of profile, which is desirable.

It was shown that dialogue duration was an important feature for usability scores [Walker et al., 1997b, Larsen, 2003, Laroche et al., 2011]: the shorter the dialogue, the higher the score. However, usability is not a strictly decreasing function of dialogue duration. It is likely to encounter dialogues with the same performance scores but different, although close, durations. Figure 8.1 shows a boxplot for the usability scores given by the users who interacted with CLASSiC System 3 according to the number of turns.

There seems to be a correlation between usability scores and dialogue duration but the figure also shows that many dialogues with different durations were rated the same. So, by forcing the system to minimise duration, one adds a constraint to the usability scores.

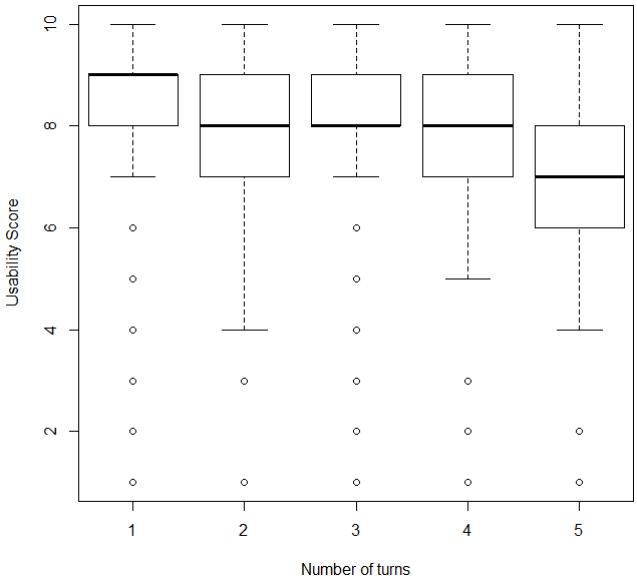


Figure 8.1: Usability scores for System 3 according to the number of turns.

8.2.5 Positioning

In this work, the performance score P is considered to be the objective function of the system. The returns at time t_0 and time t are given in Equations 8.2 and 8.3.

$$r_{t_0} = \sum_t \gamma^t R_t = P \tag{8.2}$$

$$r_t = \sum_{t' \geq t} \gamma^{t'-t} R_{t'} = \frac{P}{\gamma^t} \tag{8.3}$$

This formulation of the returns distributes the performance score among the decisions taken during a dialogue. The factor γ is chosen a little below 1, e.g., at 0.99. This is explained by the fact that the algorithms proposed in the next chapter compute intermediate rewards, distributed during the dialogue.

As said before, performance scores given by users or experts already account for dialogue length [Walker et al., 1997b, Larsen, 2003, Laroche et al., 2011]. With the objective function in Equation 8.2, dialogue length is only present in the performance score. Two algorithms computing a reward function with this interpretation are proposed and compared in the next two chapters. One of these algorithms, distance minimisation, performs a regression whereas the other, reward shaping, takes into account the policy evaluation as in IRL. It is shown that the interpretation of the performance scores that was chosen in this section outperforms the one which sees the performance score as the value of the final state. The two approaches are tested on the corpus of simulated dialogues with LTI and on simulated dialogues with the appointment scheduling simulator. Then, the importance of the quality of the representation of the state space is emphasised by an illustration on the System 3 corpus. The two algorithms and the illustration on System 3 were published first [El Asri et al., 2012], then the comparison on LTI was published [El Asri et al., 2013], and an analysis of the theoretical properties of the distance minimisation algorithms was submitted [El Asri et al., 2016a].

Chapter 9

Reward Shaping

The first algorithm which learns a reward function from a set of rated dialogues requires an estimator of the performance scores. The first part of this chapter is thus dedicated to this problem. Results of this section were published [El Asri et al., 2014a].

9.1 Dialogue Performance Estimation

Several estimators of Interaction Quality (IQ) are compared on LEGO (see Chapter 4). As said in Chapter 2, most of the models proposed to estimate either IQ or US rely on regression or classification. Here, it is proposed to apply ordinal regression, which is a classification that takes into account the order of the labels. It is shown on the LEGO corpus that ordinal regression with support vector machines performs better than several regression and classification methods on a set of value-based and behaviour-based metrics.

First, an exploratory analysis of the data in the LEGO corpus is performed. Then, the metrics used to compare the different estimators are presented. The estimators are then described and the results of the comparison on LEGO are discussed.

9.1.1 Exploratory analysis

The features in the LEGO corpus are in Table A.1 in Appendix A. This corpus contains both numerical (ASR-Confidence, #ASRRejections,...) and categorical (SDA and UDA) features. Figure 9.1 display scatter-plots to emphasise the correlation between the features in the corpus and the IQ ratings. There seems to be an obvious correlation between the IQ ratings and dialogue duration, and consequently, with all the other features correlated with dialogue duration. It is reminded here that IQ ratings on the LEGO corpus are turn-level ratings, which means that the experts gave a rating after each user-system exchange. Each dialogue started with an IQ rating equal to 5. This rating was then degraded by the expert if the quality of the dialogue had decreased. Figure 9.3 shows that actually, this apparent correlation is caused by one dialogue which is very long (1329.5s). This dialogue counts for 257 system-user exchanges that is to say 5% of the corpus of exchanges. Figures 9.1.1 and 9.1.1 reproduce the previous plots without the longest dialogue. These plots show that there is not actually an obvious correlation between IQ and dialogue duration. On the other hand, there appears to be a correlation between the IQ ratings and the number of ASR rejections and the number of re-prompts, and these two features seem to be linearly correlated. The longest dialogue should not be discarded though as it counts for 5%

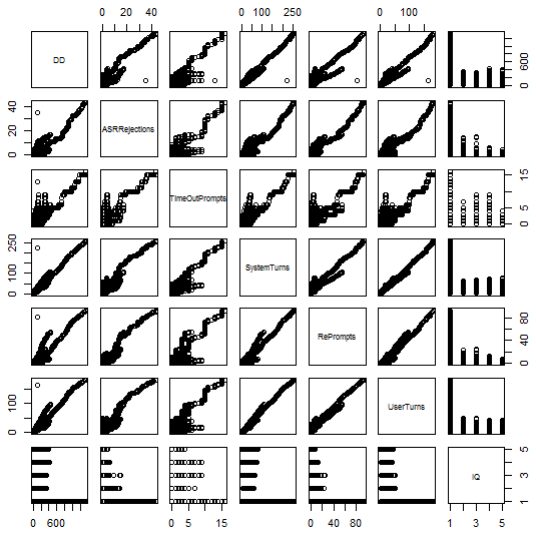


Figure 9.1: Correlation between features in LEGO and IQ ratings.

of the corpus, the estimators should determine that important features for this corpus are dialogue duration, the number of ASR rejections, and the number of re-prompts.

Before describing the regression and classification methods compared on LEGO, we describe the metrics used to perform this comparison.

9.1.2 Metrics

For RL-based systems, it is important to provide an estimator whose estimations are close to the real performance scores (either IQ or US) both in terms of values and behaviour. Following on from this consideration, it is proposed here to analyse the efficiency of the estimators based on several value-based and behaviour-based metrics: Spearman's rank correlation, average Euclidean distance, Manhattan error and Cohen's agreement coefficient κ. Note that in this particular case of turn-level ratings, if turn-level IQ predictions are used as rewards and if they maintain the ordering of system actions that was induced by the actual experts ratings, RL frameworks such as the Ordinal Markov Decision Process introduced by Weng Weng [2012] can then be applied to learn an optimal behaviour. The Manhattan error between the vector of IQ ratings y and the estimated ratings $\hat{y}$ is computed as follows:

$$\text{Manhattan\_error}(y, \hat{y}) = \frac{1}{T} \sum_{i=1}^{n} |y_i - \hat{y}_i|, \tag{9.1}$$

where T is the number of instances in the test set. Spearman's rank correlation coefficient ρ measures the correlation between the two rankings $r(y)$ and $r(\hat{y})$. Let $y = \{y_1, ..., y_n\}$ and $\hat{y} = \{\hat{y}_1, ..., \hat{y}_n\}$ be respectively the values to estimate and the vector of estimations. Let $r(y) = \{r(y_1), ..., r(y_n)\}$ and $r(\hat{y}) = \{r(\hat{y}_1), ..., r(\hat{y}_n)\}$ be their corresponding rankings. For instance, if $y = \{1, 15, 3, 12, 27\}$, then $r(y) = \{1, 4, 2, 3, 5\}$. The coefficient is comprised between -1 and 1. The closer to 1, the higher the correlation between the rankings. The

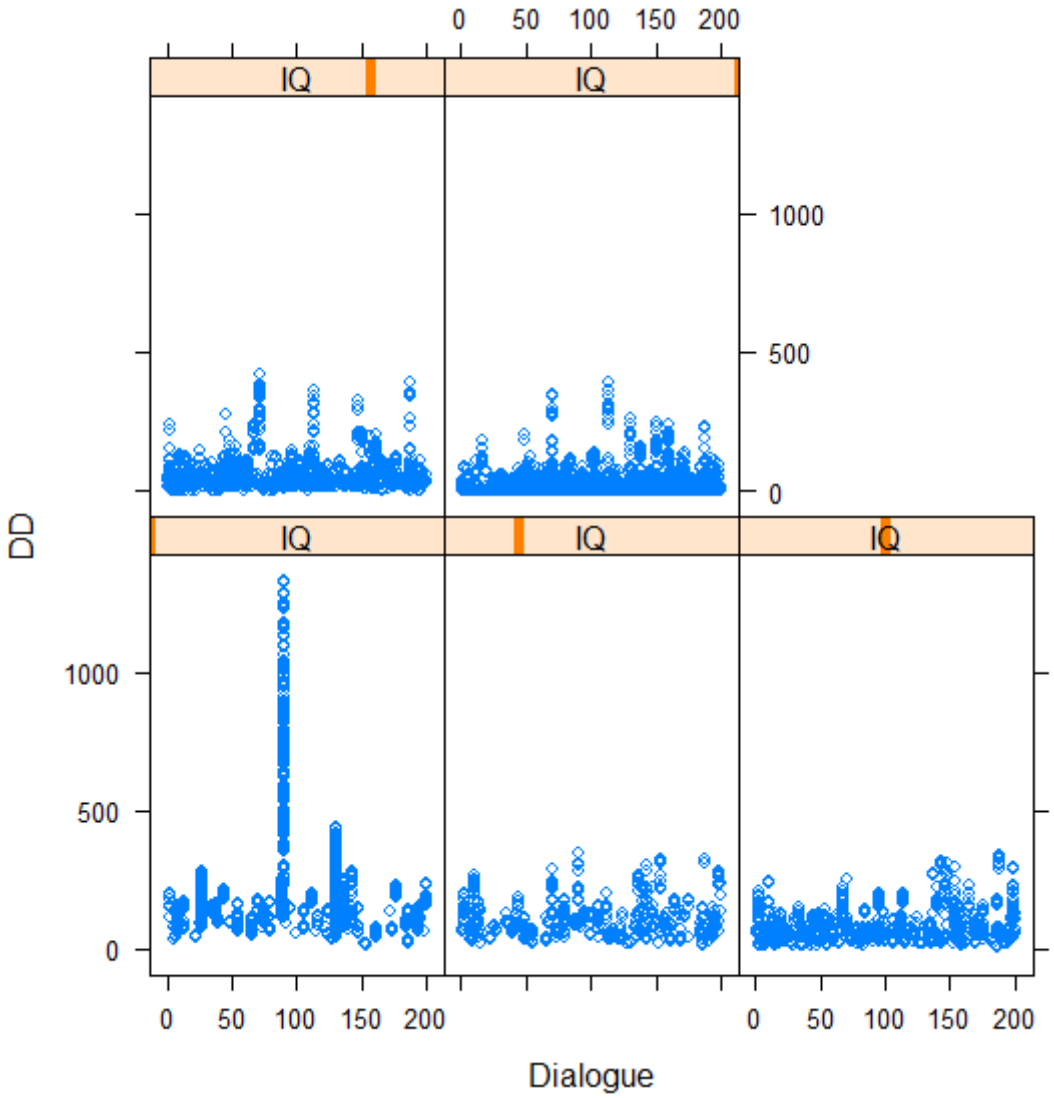


Figure 9.2: Correlation between the dialogue duration and the IQ rating (from bottom left to top right : IQ = 1, IQ = 2, IQ = 3, IQ = 4, IQ = 5).

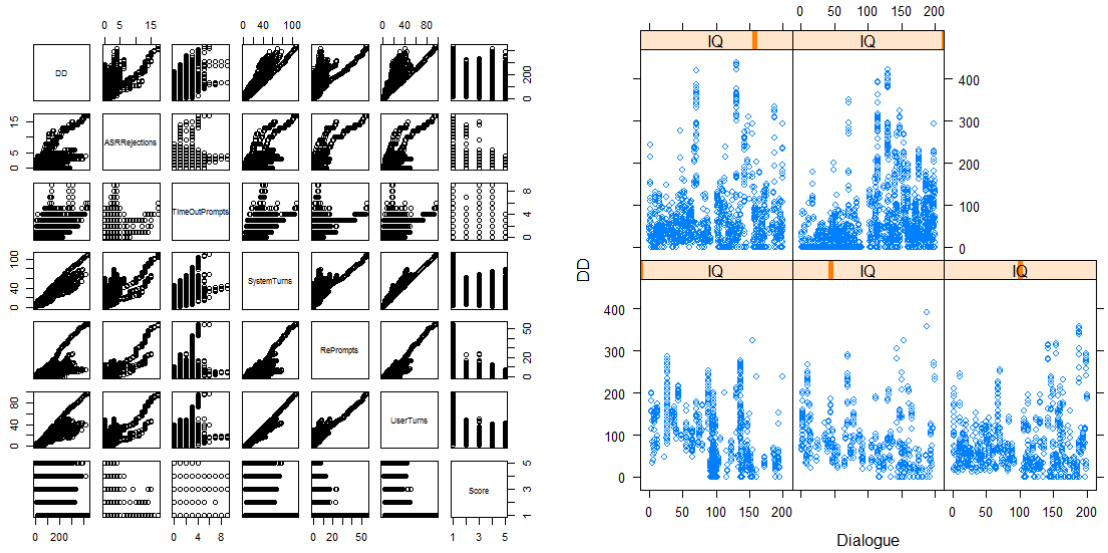


Figure 9.3: Analysis of LEGO without the longest dialogue. On the left: correlation between features in LEGO and IQ ratings. On the right: correlation between the dialogue duration and the IQ rating.

coefficient is computed as follows:

$$\rho(y,\hat{y}) = \frac{\sum_i (r(y_i) - \text{mean}(r(y)))\sum_i (r(\hat{y}_i) - \text{mean}(r(\hat{y})))}{\sqrt{\sum_i (r(y_i) - \text{mean}(r(y)))^2}\sqrt{\sum_i (r(\hat{y}_i) - \text{mean}(r(\hat{y})))^2}}. \tag{9.2}$$

The Euclidean error is computed as such:

$$E(y,\hat{y}) = \left(\frac{1}{n}\sum_{i=1}^{n}(y^i - \hat{y}^i)^2\right)^{\frac{1}{2}}. \tag{9.3}$$

The agreement coefficient is computed according to Equation 3.1. In the following section, the regression and classification-based IQ estimators are described. These methods were applied to the LEGO corpus and evaluated with cross validation: the corpus was split into 10 sets among which each had 90% of the data. For each set, the remaining 10% served as a test corpus. The metrics were computed and averaged on the 10 test sets.

9.1.3 Models

Multiple Linear Regression (MLR)

MLR is sensitive to correlated features. Indeed, the method might select features which are correlated and the interpretation of the resulting model might be biased by this phenomenon. To avoid this drawback, features

were selected one by one. Single linear regression was performed for each feature and the feature with the highest coefficient in absolute value was kept[1]. Let us denote by f^i the most explanatory feature and β^i its linear regression coefficient. The error $IQ - \beta^i f^i$ then replaced IQ as objective value and the single regression coefficients for the remaining features were computed. This process was repeated until the squared error on the test sets was minimised. The methodology is given in Algorithm 3.

Algorithm 3 Multiple Linear Regression

Inputs:
The training corpus of evaluated user-system exchanges $(x_j, y_j)_{j=1,..,n}$. $\forall\ j,\ x_j = (f_j^1, ..., f_j^m)$.
Initialise the model $\text{model}_0 = 0$.
Initialise the set of selected features $F = \emptyset$.
Initialise the objective vector : $o_0 = y$.
repeat
    Perform m linear regressions $o_k = \beta_1^i f^i + \beta_0^i$, where $f^i = (f_1^i, ..., f_n^i), i = 1, .., m$.
    Select f^i such that $\beta_1^i = \max_j \beta_1^j$.
    $o_k = o_{k-1} - \beta_1^i f^i - \beta_0^i$
    $F = F \cup \{f^i\}$
    Perform MLR with the features in F, $\text{model}_k = \text{model}_{k-1} + \sum_j \beta^j f^j + \beta^0$.
    Apply model_k to the test set and compute the average Euclidean error Euclidean_error_k.
until $\text{Euclidean\_error}_k > \text{Euclidean\_error}_{k-1}$

The features which were always selected first by the algorithm were (#)ASRRejections, WPST, WPUT, DD and UDA_REQUEST_HELP.

Multivariate Adaptive Regression Splines (MARS)

MARS was introduced by Friedman [Friedman, 1991]. MARS is indicated when the linear correlation between the objective function and the features is piecewise. It models IQ as a linear function of hinge functions of the form: $B_j(x) = max(0, x - s)$ or $B_j(x) = max(0, s - x)$ where s is a given training sample. MARS was run with the ARES toolbox for Matlab[2]. The algorithm requires two parameters, the maximum number of hinge functions *maxFuncs* and *cubic*, which indicates whether cubic splines should be used to smooth the edges. Different combinations of these parameters were tried but no significant impact on the evaluation metrics was noticed. Hence, the default parameters (recommended in [Friedman, 1991]) were kept, maxFuncs was set to 21 and cubic splines were not used. The main features that were selected were (#)RePrompts, (Mean)ASRConfidence, #RePrompts, (#)TimeOutPrompts, %RePrompts.

Gaussian Processes

A Gaussian Process (GP) is a set of random variables which joint distribution is Gaussian [Rasmussen and Williams, 2005]. To perform GP regression, y is modelled as: $y = \hat{y} + \Delta\hat{y}$ with $\Delta\hat{y}$ a centered Gaussian noise vector and $\hat{y}$ is a GP which mean and variance are to be determined. One advantage in using GP

[1] it is recalled that features were first centered and normalised

[2] Jekabsons G., ARESLab: Adaptive Regression Splines toolbox for Matlab/Octave, 2011, available at http://www.cs.rtu.lv/jekabsons/

Covariance type	$k(x, x')$
Linear	$x^T \Lambda^{-2} x'$
Rational quadratic	$\sigma^2(1 + \frac{1}{2\alpha}(x - x')^T \Lambda^{-2}(x - x'))^{-\alpha}$
Squared Exponential	$\exp(-\frac{1}{2}(x - x')^T \Lambda^{-2}(x - x'))$

Table 9.1: Covariance matrices for Gaussian processes regression.

over other regression methods is that it returns an entire distribution over the possible values instead of a point-based decision rule. Another advantage is that it enables a Bayesian treatment of the data, inferring a posterior distribution from a prior belief and observations. Following Rasmussen and Williams [Rasmussen and Williams, 2005], the prior was a centered distribution $GP(0, k(x, x))$ with k the covariance function. Exact inference could then be performed from the training samples to compute the posterior distribution. The GPML Matlab library by Rasmussen and Nickisch[3] was used to do GP-regression. Three covariance functions were tested: linear, rational quadratic and squared exponential. Their expressions are recalled in Table 9.1. The parameters Λ, σ and α were all learnt by automatic relevance determination [Jung and Stone, 2009]. Since this computation is very costly ($O(N^3)$ with N the number of training samples), only a dictionary of points was kept in memory, following Engel [Engel et al., 2005]. This reduced computational load to $O(m^2 n)$ with m the dictionary size. Section 9.1.4 only presents the results with the squared exponential kernel as it was the best performing according to the metrics defined in Section 9.1.2.

Support Vector Machines (SVM)

Originally, SVM [Vapnik, 1998] were meant for 2-class classification. SVM take a set of labelled examples $\{(x_j, y_j)\}$ where $y_j \in \{-1, 1\}$ and then they look for a hyperplane in a feature space that separates the two classes so that the minimal margin between the samples and the hyperplane is maximised. The decision function returned is given in Equation 9.4 where $\phi(x)$ is the projection of x onto the high dimensional feature space.

$$f(x) = \text{sign}(\omega^T \phi(x) + b) \tag{9.4}$$

SVM for Classification (SVMClass), Support Vector Machines for Regression (SVR) and Support Vector Machines for Ordinal Regression (SVOR) solve a quadratic programming problem to find the weight vector ω and the constant b. The differences between the approaches lie in the constraints of the problem and the form of the decision function. These are detailed in the following sections.

Support Vector Machines for Classification To perform 5-class classification with SVM, a one-versus-one approach was applied. 10 classifiers were learnt, each deciding between two classes. A new example was assigned the class that was the most voted for. Ties always favoured the lowest label as predicting a low IQ instead of a higher one is more critical than the other way round. The problem solved by each classifier is recalled in Equation 9.5. The decision function is the one of Equation 9.4.

[3] http://wwww.gaussianprocesses.org/gpml

Kernel function type	$k(x, x')$
Linear	$x^T x'$
Radial Basis Function	$\exp(-\gamma(x - x')^T(x - x'))$

Table 9.2: Kernel functions for SVMClass. In our experiment, γ was set to $\frac{1}{67}$, 67 being the number of features per sample point.

	Manhattan error	Euclidean error	Spearman's ρ	Cohen's κ
MLR	0.600 ± 0.007	0.792 ± 0.009	0.831 ± 0.004	0.364 ± 0.008
MARS	0.608 ± 0.008	0.799 ± 0.01	0.821 ± 0.002	0.368 ± 0.009
GP	0.593 ± 0.007	0.853 ± 0.01	0.833 ± 0.005	0.408 ± 0.006
SVR	0.559 ± 0.007	$\mathbf{0.766 \pm 0.009}$	0.837 ± 0.004	0.414 ± 0.008
SVOR	$\mathbf{0.453 \pm 0.005}$	$\mathbf{0.763 \pm 0.008}$	$\mathbf{0.842 \pm 0.005}$	$\mathbf{0.489 \pm 0.006}$
SVMClass	0.580 ± 0.009	0.965 ± 0.01	0.776 ± 0.004	0.421 ± 0.005

Table 9.3: Results of the 10-fold cross validation on the LEGO corpus. 95% confidence interval bounds are provided for each metric.

$$\begin{aligned} &\min_{\omega,b,\xi} \frac{1}{2}\omega^T\omega + \sum_i \xi_i \\ &\text{subject to } y_i(\omega^T\phi(x_i) + b) > 1 - \xi_i \\ &\xi_i > 0 \,\forall\, i \end{aligned} \tag{9.5}$$

The LIBSVM software [Chang and Lin, 2011] was used for SVMClass. As showed in Table 9.2, two kernel functions were tested, a linear and a radial basis function. We only present the results with the radial basis kernel as it performed better than the others.

Support Vector Machines for Regression Drucker *et. al* [Drucker et al., 1996] introduced SVR. The weights of the SVM are learnt to assure that the absolute value error on each sample is less than a parameter ϵ as shown in Equation 9.6.

$$\begin{aligned} &\min_{\omega,b,\xi,\xi^*} \frac{1}{2}\omega^T\omega + \sum_i \xi_i + \sum_i \xi_i^* \\ &\text{subject to } \omega^T\phi(x_i) + b - y_i < \epsilon + \xi_i \\ &\qquad y_i - \omega^T\phi(x_i) - b < \epsilon + \xi_i^* \\ &\qquad \xi_i, \xi_i^* > 0 \,\forall\, i \end{aligned} \tag{9.6}$$

The prediction for a new example x is $f(x) = \omega^T x + b$. LIBSVM was used once again for SVR with a radial basis kernel. The parameter ϵ was set to 0.1.

Support Vector Machines for Ordinal Regression Ordinal regression or ranking learning aims to build a model to predict ordinal ranks. SVOR [Chu and Keerthi, 2007] adds constraints so that the ordering between the scores are accounted for. The problem solved is in Equation 9.7 where j goes from 1 to $r-1$ and $C > 0$. The idea is to find an optimal weight vector ω as in SVMClass but also $r-1$ thresholds b_j defining parallel discriminant hyperplanes for the r labels. In short, for each example x_i^j (the upperscript j means that x_i^j belongs to the j-th category), the function value $\omega^T \phi(x_i^j)$ should be lower than the lower margin $(b_j - 1)$ and each example x_i^{j+1} should have a function value $\omega^T \phi(x_i^{j+1})$ higher than the upper margin $(b_{j+1} - 1)$. More details about this algorithm can be found in [Chu and Keerthi, 2007].

$$
\begin{aligned}
\min_{\omega,b,\xi,\xi^*} \; & \frac{1}{2}\omega^T\omega + C\sum_{j=1}^{r-1}\left(\sum_{i=1}^{n^j}\xi_i^j + \sum_{i=1}^{n^{j+1}}\xi_i^{*j+1}\right) \\
\text{subject to } & \omega^T\phi(x_i^j) - b_j \leq -1 + \xi_i^j \\
& \xi_i^j \geq 0, \text{ for } i = 1...n^j \\
& \omega^T\phi(x_i^{j+1}) - b_j \geq 1 - \xi_i^{*j+1} \\
& \xi_i^{*j+1} \geq 0, \text{ for } i = 1...n^{j+1} \\
& b_{j-1} \leq b_j \text{ for } j = 2, ..., r-1
\end{aligned} \tag{9.7}
$$

A new example x is given the label $\text{argmin}_i\{i \mid \omega^T\phi(x) < b_i\}$. We computed SVOR with a Gaussian kernel using the C program available at http://www.gatsby.ucl.ac.uk/chuwei/svor.htm.

9.1.4 Results

The results of the test are given in Table 9.3. For the regression techniques, Cohen's κ coefficient was computed on the rounded predictions made by the models.

Surprisingly, MLR performed quite well on the corpus. Schmitt *et. al* applied MLR to the same corpus and after rounding the returned values, they obtained a κ of 0.35 and a ρ of only 0.46. When we round the values returned by MLR, we obtain a ρ equal to 0.805. This difference of results should be explained by the features sets used in both cases. We also believe splitting categorical features into a set of boolean variables is responsible for this difference. GP performed better than MLR and MARS except for the Euclidean error. Since MLR and MARS minimise this error while GP does not, this result could be expected. SVR, on the other hand, performed better than the previously mentioned techniques on all metrics. A significant improvement can be observed in particular concerning the Euclidean and Manhattan errors. The results of SVR are quite good because the constraints on the function value inferred by SVR imposed a very low absolute error on the samples.

SVOR outperforms all these methods on each metric. A significant improvement is done on the Manhattan error and Cohen's κ. The Manhattan error is improved of 0.1 point compared to SVR. Cohen's κ is improved of 0.06 point compared to SVMClass.

SVMClass has the lowest ranking correlation coefficient. This supports the fact that classification techniques are not the most appropriate for IQ prediction because they ignore the constraint imposed by the order of the labels. Nevertheless, as expected SVMClass has the second highest κ coefficient behind SVOR. The

classification performances of SVMClass and SVOR can be further compared thanks to their confusion matrices (Table 9.4). The results predicted by SVOR deviated less from the real values than the ones predicted by SVMClass. For instance, a true 1 was predicted as a 3, a 4 or a 5 88 times with SVOR against 236 times with SVMClass. The recall and precision for critical values 1 and 2 are also significantly higher with SVOR.

SVOR	T 1	T 2	T 3	T 4	T 5	Precision
P 1	572	99	32	4	1	0.808
P 2	124	145	118	33	1	0.344
P 3	69	236	378	228	23	0.405
P 4	18	76	347	716	359	0.472
P 5	1	1	20	269	1410	0.829
Recall	0.730	0.260	0.422	0.573	0.786	
SVMC	T 1	T 2	T 3	T 4	T 5	Precision
P 1	541	95	55	27	3	0.719
P 2	57	70	55	23	2	0.183
P 3	98	188	253	164	19	0.313
P 4	81	182	432	626	274	0.440
P 5	7	22	100	410	1496	0.781
Recall	0.690	0.126	0.282	0.501	0.834	

Table 9.4: Confusion matrices for SVOR and SVMClass. T i means that i is the true value and P i means i is the predicted value.

This section has proposed an estimator for user satisfaction and showed that this estimator, Support Vector Ordinal Regression (SVOR), performed better than many other estimators on a number of metrics. As said in chapter 5, the user satisfaction scores for the appointment scheduling simulator are estimated with SVOR. The SVOR estimator was trained on the DINASTI corpus, to estimate user satisfaction from user task completion, number of dialogue turns, average ASR score, and number of ASR rejections. In the next section, SVOR is used with the first reward learning algorithm, named *reward shaping*.

9.2 Reward Shaping

It is recalled here that the two reward learning algorithms, Reward Shaping (RS) and Distance Minimisation (DiM), are to be applied to a corpus of dialogues collected with an RL-based system. For clarity of the presentation, it is supposed that the learning framework is MDP. The two algorithms are also illustrated on the MVDP-based System 3 in Section 10.4.

9.2.1 Preliminaries

The DM is thus an MDP $M = (S, A, R, P, \gamma)$. R is the spoken dialogue system's original reward function, handcrafted by the SDS designer. This reward function might have served to guide the system's behaviour during corpus collection. However, as said in the introduction, it is advised to follow a random policy during these dialogues so that the algorithms can exploit many samples for each state-action pair. The first step

after the corpus $\mathcal{D}$ has been collected with the MDP$\backslash R$ $M\backslash R = (S, A, P, \gamma)$ is to learn a state space representation $\tilde{S}$ which adds numerical dialogue parameters (dialogue duration, etc.) to the information states S. In Chapter 7, a memory-efficient representation was proposed. For simplicity of the presentation, in this chapter, a grid-based representation will be used. The state space $\tilde{S}$ is thus composed of the states of the grid. For the appointment scheduling simulator, a state is for instance: {System Task Completion ≤ 0.7, last UDA = REFUSE_AND_ASK, Dialogue Duration ≤ 3.0, Average ASR Score > 0.75}. Let us define $\tilde{M} = (\tilde{S}, A, \tilde{R}, \tilde{P}, \gamma)$. During a dialogue D^i (whose first and final dialogue turns are respectively $t_0 = 0$ and t^i_f), for each transition $(\tilde{s}, \tilde{s}')$, the reward function $\tilde{R}$ gives the reward:

$$\tilde{R} : (\tilde{s}, \tilde{s}') \in \tilde{S} \times \tilde{S} \mapsto \begin{cases} 0 & \text{if } \tilde{s}' \text{ is not the final state of the dialogue,} \\ \gamma^{\frac{P^i}{t^i_f}} & \text{otherwise.} \end{cases} \tag{9.8}$$

The reward function does not depend on the current action a, only on the states $\tilde{s}$ and $\tilde{s}'$. As a consequence, to simplify the notation, the rewards are defined on state-state transitions $(\tilde{s}, \tilde{s}')$ rather than state-action-state transitions $(\tilde{s}, a, \tilde{s}')$. From $\tilde{M}$ and the corpus of dialogues $\mathcal{D}$, the reward shaping algorithm computes a new reward function $\hat{R}$ and returns $\hat{M} = (\tilde{S}, A, \hat{R}, \tilde{P}, \gamma)$. The reward function $\hat{R}$ is defined on the state space $\tilde{S}$.

In all that follows the value functions Q and V (respectively $\tilde{Q}$ and $\tilde{V}$) refer to $\hat{M}$ (respectively $\tilde{M}$). Before describing RS, the underlying theory of reward shaping is presented.

9.2.2 The Reward Shaping Theory

Ng et al. [1999] showed, in the context of MDP-based RL, that adding a potential-based function F to a reward function R_0 did not change the optimal policy. An optimal policy for the function

$$R_0 + F : (s, a, s') \in S \times A \times S \mapsto R_0(s, a, s') + \gamma F(s') - F(s) \tag{9.9}$$

is also optimal for R_0. Proof that an optimal policy for $R_0 + F$ is also optimal for R_0 is reproduced here from [Ng et al., 1999]. Let us suppose that γ is different than 1. Let us consider two MDP $M = (S, A, R_0, P, \gamma)$ and $M' = (S, A, R_0 + F, P, \gamma)$. The optimal Q-function Q^*_M satisfies the Bellman optimality equation:

$$Q^*_M(s, a) = E_{s' \sim P_a(s,.)}[R_0(s, a, s') + \gamma \max_{a' \in A} Q^*_M(s', a')].$$

It can be deduced that:

$$Q^*_M(s, a) - F(s) = E_{s' \sim P_a(s,.)}[R_0(s, a, s') + \gamma F(s') - F(s) + \gamma(\max_{a' \in A} Q^*_M(s', a') - F(s'))].$$

Let us define $\hat{Q}_{M'}(s, a) = Q^*_M(s, a) - F(s)$. Then,

$$\hat{Q}_{M'}(s, a) = E_{s' \sim P_a(s,.)}[R_0(s, a, s') + \gamma F(s') - F(s) + \gamma \max_{a' \in A} \hat{Q}_{M'}(s', a')].$$

This is precisely the Bellman optimality equation for M'. Thereby, $Q^*_{M'}(s, a) = \hat{Q}_{M'}(s, a) = Q^*_M(s, a) - F(s)$. An optimal policy for M' satisfies:

$$\begin{aligned} \pi^*_{M'}(s) &\in \operatorname*{argmax}_{a \in A} Q^*_{M'}(s, a) \\ &= \operatorname*{argmax}_{a \in A} Q^*_M(s, a) - F(s) \\ &= \operatorname*{argmax}_{a \in A} Q^*_M(s, a). \end{aligned} \tag{9.10}$$

The policy $\pi^*_{M'}$ is thus also optimal for M. Ng et al. pushed the approach further by trying to find a good fit for the potential function. The following equation holds for the optimal value functions of M and M':

$$V^*_{M'}(s) = V^*_M(s) - F(s).$$

If, for every state s, the potential $F(s)$ is equal to $V^*_M(s)$ then $V^*_{M'}(s)$ is equal to 0. The corresponding Q-function is particularly easy to learn since it only requires learning the non null values, which correspond to the non optimal actions. Indeed, in this case:

$$\begin{aligned} Q^*_{M'}(s,a) &= Q^*_M(s,a) - F(s) \\ &= Q^*_M(s,a) - V^*_M(s). \end{aligned} \tag{9.11}$$

Consequently, if $a \in \pi^*(s)$, then $Q^*_{M'}(s,a)$ is equal to 0. The agent must thus only learn the values of the Q-function that are not null. To illustrate the fact that an estimation of V^*_M is an appropriate potential function to speed up learning, Ng et al. applied reward shaping to a 10×10 grid-world. This task consists of learning to reach a corner on a two-dimensional grid starting from the opposite corner. Similarly to mountain car, the aim is to find the shortest path to the goal. Thereby, at each time step, the agent receives a reward equal to -1 and a reward equal to 0 once the goal position is reached. The agent's actions are the four compass directions with added noise[4]. The authors learnt a policy with reward shaping by using an estimation of $V^*_M(s)$ for intermediate rewards. They showed that the time needed to learn an optimal policy was significantly reduced compared to the case where reward shaping was not applied. The authors also illustrated reward shaping on a grid-world task with several sub-goals: the agent needed to go through certain cells before reaching the goal cell. For this task, the authors did not define the potential function as an estimation of the optimal value function but instead, they defined $F(s)$ depending on the sub-goals which have been reached before reaching s. Ng et al. showed that learning with this function was significantly sped up compared to learning without shaping and that it was also faster compared to learning with the shaping function which gave an estimation of $V^*_M(s)$. It is proposed in the following section to learn a potential function which is inspired by this approach in that it accounts for the current progress in the dialogue task.

9.2.3 Application to the Reward Inference Problem

The reward shaping algorithm proposed here models the reward function as the potential-based function $\hat{R}(\tilde{s}, \tilde{s}') = \gamma F(\tilde{s}') - F(\tilde{s})$.

In this section, it is shown that an optimal policy for this reward function is also optimal for the function that distributes a final reward equal to $\gamma F(\tilde{s}_{t_f})$ and intermediate rewards equal to 0. In the dialogue management context, the learning task is episodic so there exists a set S_f of final states. This set is distinct from S. If necessary, final states are added to the original set proposed by the designer, e.g., a final state corresponding to the user's hanging up. This will be further discussed in Section 10.5. Under this assumption, the optimality Bellman equations for $\hat{M}$ at a state $\tilde{s}$ can be written as:

$$\begin{aligned} Q^*_{\hat{M}}(\tilde{s},a) &= E_{\tilde{s}' \in \tilde{S} \sim P_a(\tilde{s},.)}[\gamma F(\tilde{s}') - F(\tilde{s}) + \gamma \max_{a' \in A} Q^*_{\hat{M}}(\tilde{s}', a')] + E_{\tilde{s}' \in \tilde{S}_f \sim P_a(\tilde{s},.)}[\gamma F(\tilde{s}') - F(\tilde{s})] \\ &= E_{\tilde{s}' \sim P_a(\tilde{s},.)}[\delta_{\tilde{s}' \in \tilde{S}_f}(\gamma F(\tilde{s}') - F(\tilde{s})) + \delta_{\tilde{s}' \notin \tilde{S}_f}(\gamma F(\tilde{s}') - F(\tilde{s}) + \gamma \max_{a' \in A} Q^*_{\hat{M}}(\tilde{s}', a'))]. \end{aligned} \tag{9.12}$$

[4] 80% of the time the agent moves in the intended direction and 20% of the time, it moves in a random direction.

From this equation, it can be deduced that:

$$Q^*_{\hat{M}}(\tilde{s},a) + F(\tilde{s}) = E_{\tilde{s}'\sim P_a(\tilde{s},.)}[\delta_{\tilde{s}'\in\tilde{S}_f}\gamma F(\tilde{s}') + \delta_{\tilde{s}'\notin\tilde{S}_f}\gamma(\max_{a'\in A} Q^*_{\hat{M}}(\tilde{s}',a') + F(\tilde{s}'))]. \tag{9.13}$$

Now let us define $\hat{Q}_{\tilde{M}'}(\tilde{s},a) = Q^*_{\hat{M}}(\tilde{s},a) + F(\tilde{s})$, where $\tilde{M}' = (\tilde{S}, A, R_0, \tilde{P}, \gamma)$ and R_0 is defined as follows:

$$R_0 : (\tilde{s},a,\tilde{s}') \mapsto \delta_{\tilde{s}'=\tilde{s}_{t_f}}\gamma F(\tilde{s}_{t_f}). \tag{9.14}$$

Then,

$$\hat{Q}_{\tilde{M}'}(\tilde{s},a) = E_{\tilde{s}'\sim P_a(\tilde{s},.)}[\delta_{\tilde{s}'\in\tilde{S}_f}\gamma F(\tilde{s}') + \delta_{\tilde{s}'\notin\tilde{S}_f}\gamma \max_{a'\in A}\hat{Q}_{\tilde{M}'}(\tilde{s}',a')]. \tag{9.15}$$

Equation 9.15 can be re-written as:

$$\hat{Q}_{\tilde{M}'}(\tilde{s},a) = E_{\tilde{s}'\sim P_a(\tilde{s},.)}[R_0(\tilde{s},a,\tilde{s}') + \delta_{\tilde{s}'\notin\tilde{S}_f}\gamma \max_{a'\in A}\hat{Q}_{\tilde{M}'}(\tilde{s}',a')]. \tag{9.16}$$

This equation is the Bellman optimality equation for $\tilde{M}'$ so, as in the previous demonstration, a policy optimal for R_0 is also optimal for $\hat{R}$.

Whence, the reward function computed by the reward shaping algorithm allows to learn a policy which is also optimal for a function which gives a final reward equal to $\gamma F(\tilde{s}_{t_f})$ and intermediate rewards equal to 0. This can also be showed by computing the returns corresponding to the function $\hat{R}$:

$$\begin{aligned}
r_0 &= \sum_{t\geq 0}\gamma^t\hat{R}(\tilde{s}_t,\tilde{s}_{t+1}) = \sum_{t=0}^{t_f-1}\gamma^t(\gamma F(\tilde{s}_{t+1}) - F(\tilde{s}_t)) \\
&= \sum_{t=0}^{t_f-1}\gamma^{t+1}F(\tilde{s}_{t+1}) - \sum_{t=0}^{t_f-1}\gamma^t F(\tilde{s}_t) \\
&= \sum_{t=1}^{t_f}\gamma^t F(\tilde{s}_t) - \sum_{t=1}^{t_f-1}\gamma^t F(\tilde{s}_t) - F(\tilde{s}_{t_0}) \\
&= \sum_{t=1}^{t_f-1}\gamma^t F(\tilde{s}_t) + \gamma^{t_f}F(\tilde{s}_{t_f}) - \sum_{t=1}^{t_f-1}\gamma^t F(\tilde{s}_t) - F(\tilde{s}_{t_0}) \\
&= \gamma^{t_f}F(\tilde{s}_{t_f}) - F(\tilde{s}_{t_0}).
\end{aligned} \tag{9.17}$$

Similar calculation leads to the following equation for the other returns: $\forall\, t > 0, r_t = \gamma^{t_f-t}F(\tilde{s}_{t_f}) - F(\tilde{s}_t)$. With this model of rewards, the optimisation of the Q-function only depends on the final state s_{t_f}[5].

As previously, the following equality holds concerning $V^*_{\tilde{M}'}$ and $V^*_{\hat{M}}$:

$$V^*_{\hat{M}} = V^*_{\tilde{M}'} - F. \tag{9.18}$$

[5] since the other part of the return $F(\tilde{s}_t)$ does not depend on the action chosen at time t.

Thereby, defining F as an estimation of $V^*_{\tilde{M}'}$ will amount to learning the non-null values of the Q-function of $\hat{M}$. Such a function F should satisfy, for all non-terminal state $\tilde{s}$:

$$F(\tilde{s}) = E_{\pi^*_{\tilde{M}'}}[\gamma^{t_f - t} F(\tilde{s}_{t_f}) \mid \tilde{s}]. \tag{9.19}$$

In other terms, the potential function should estimate, at each information state $\tilde{s}$, the expected value of the final information-feature state according to F and an optimal policy for $\tilde{M}'$ (and $\hat{M}$) π^*. Following the interpretation of the performance scores suggested in Chapter 8, given a dialogue D^i and the reward function R_0 defined in Equation 9.14, the return at time t should be equal to P^i/γ^t:

$$\begin{aligned} r^i_{0,t} &= \sum_{t' \geq t} \gamma^{t'-t} R_0(\tilde{s}_{t'}, a_{t'}, \tilde{s}_{t'+1}) \\ &= \gamma^{t^i_f - t} F(\tilde{s}_{t^i_f}) \\ &= \frac{P^i}{\gamma^t}. \end{aligned} \tag{9.20}$$

As a consequence, it is required that:

$$F(\tilde{s}_{t^i_f}) = \frac{P^i}{\gamma^{t^i_f}} \tag{9.21}$$

$$\begin{aligned} \text{and, for all non-terminal state } \tilde{s},\ F(\tilde{s}) &= E_{\pi^*}\left[\gamma^{t_f - t} \frac{P}{\gamma^{t_f}} \mid \tilde{s}_t = \tilde{s}\right] \\ &= E_{\pi^*}\left[\frac{P}{\gamma^t} \mid \tilde{s}_t = \tilde{s}\right]. \end{aligned} \tag{9.22}$$

Equation 9.21 implies that the value of the performance score P^i is available for each dialogue D^i. However, the third guideline stated in the introduction stipulates that the reward function can be used online and the scores P^i are only available for the evaluated dialogues. Thereby, it is necessary to build an estimator $\hat{P}$ of P to define the value of the potential function at the final dialogue states. As a result, in order to define F as an estimation of $V^*_{M'}$, the function F should be defined as follows:

$$F(\tilde{s}_{t^i_f}) = \frac{\hat{P}^i}{\gamma^{t^i_f}} \tag{9.23}$$

$$\text{and, for all non-terminal state } \tilde{s},\ F(\tilde{s}) = E_{\pi^*}\left[\frac{\hat{P}}{\gamma^t} \mid \tilde{s}_t = \tilde{s}\right]. \tag{9.24}$$

However, here, the function F is defined as:

$$F(\tilde{s}_{t^i_f}) = \frac{\hat{P}^i}{\gamma^{t^i_f}} \tag{9.25}$$

$$\text{and, for all non-terminal state } \tilde{s},\ F(\tilde{s}) = E_{\pi^*}\left[\frac{P}{\gamma^t} \mid \tilde{s}_t = \tilde{s}\right]. \tag{9.26}$$

The function $V^*_{\hat{M}}$ learnt by the reward shaping algorithm is thus:

$$V^*_{\hat{M}}(\tilde{s}) = E_{\pi^*}\left[\frac{\hat{P}^i}{\gamma^t} \mid \tilde{s}_t = \tilde{s}\right] - E_{\pi^*}\left[\frac{P}{\gamma^t} \mid \tilde{s}_t = \tilde{s}\right], \tag{9.27}$$

and the reward function is

$$\hat{R} : (\tilde{s}_0, \tilde{s}') \mapsto \begin{cases} \gamma E_{\pi^*}\left[\frac{P}{\gamma^t} \mid \tilde{s}_t = \tilde{s}'\right] & \text{if } \tilde{s}' \text{ is not the final state of the dialogue,} \\ \gamma \frac{\hat{P}^i}{\gamma^{t^i_f}} & \text{otherwise.} \end{cases} \tag{9.28}$$

The definition of F given in Equations 9.21 and 9.22 requires learning a policy π^* which would be optimal for $\tilde{M}'$. As it was shown in this section, an optimal policy for $\hat{M}$ is also optimal for $\tilde{M}'$. The algorithm proposed in the next section takes advantage of this property to define a stopping criterion based on the computation of an optimal policy for $\hat{M}$ and the estimation of F at each state.

9.2.4 Description of the Reward Shaping Algorithm

The reward shaping process is described in Algorithm 4. The algorithm starts with a random policy π_0 and a null Q-function Q^{π_0}. During the first step of the algorithm, $\hat{R}$ is initialised to be equal to $\tilde{R}$ (see Equation 9.8). Every transition $(\tilde{s}, \tilde{s}')$ observed in the corpus is mapped to the reward $\tilde{R}(\tilde{s}, \tilde{s}')$. The Q-function and the policy are then updated according to these rewards. The resulting policy is π_1. Then, the $\tilde{V}$-values are updated according to π_1. The function $\hat{R}$ is then updated according to $\tilde{V}$ and defined according to Equation 9.28.

Algorithm 4 necessitates to update $\tilde{V}$ and Q given a batch of episodes and a reward function ($\tilde{R}$ for $\tilde{V}$ and $\hat{R}$ for Q). To do these computations, batch RL algorithms such as fitted value iterations and FQI can be used (see Chapter 7). The algorithm stops when the l_2 norm of the difference between the last $\tilde{V}$-function computed by the algorithm ($\tilde{V}^{\pi_k}$) and the one computed at the previous iteration ($\tilde{V}^{\pi_{k-1}}$) is below a given threshold ϵ. At each iteration k of the algorithm, the value to be optimised at each state $\tilde{s}_t$ is the same as the one that was to be optimised at the previous iteration $k-1$, i.e, $\gamma^{-t}\hat{P}$. The convergence to an optimal policy for $\hat{M}$ thus depends on the convergence of the learning algorithm used to update the Q-function.

9.2.5 Comparison of Different Approaches on the Appointment Scheduling Simulator

Here, two hypotheses made earlier in this chapter are tested on the appointment scheduling simulator, namely the assumption that it is preferable to define F at non-terminal states s_t as $E_{\pi^*}\left[\frac{P}{\gamma^t} \mid \tilde{s}_t = \tilde{s}'\right]$ rather than $E_{\pi^*}\left[\frac{\hat{P}}{\gamma^t} \mid \tilde{s}_t = \tilde{s}'\right]$ and the hypothesis according to which the final reward should be defined as P/γ^t and not P. The functions tested on the corpus are listed in Table 9.5. Another function is compared to these: the one which gives the performance score at the end of the dialogue and 0 everywhere else. This function is denoted by R_{PS} on Figure 9.4. These functions are tested according to the following protocol. On three sizes of training corpora: 500, 1000 and 2000, the reward shaping algorithm is applied with each function and then new dialogues are simulated with these reward functions and a policy is learnt with SARSA(λ) and an ϵ-greedy exploration with ϵ set to 0.01 (see chapter 5). The policies are then tested on 100 dialogues after every one of the first 50 dialogues and then after every 100 dialogues.

The estimator $\hat{P}$ is supposed to be an estimation of the objective function. In order to reproduce this condition, a Gaussian noise of variance 1 was applied to the user satisfaction scores in the DINASTI corpus

Algorithm 4 Reward shaping algorithm

Inputs:
The evaluated dialogues $D^1, ..., D^N$ with performance scores $P^1, ..., P^N$; an MDP$\backslash R$ $M\backslash R$; a stopping criterion ϵ
Define:
A starting random policy π_0, $\forall\ (\tilde{s}, a), Q^{\pi_0}(\tilde{s}, a) = 0$.

Initialisation:
for all $D^i \in D^1, .., D^N$ **do**
  for all transition $(\tilde{s}, \tilde{s}') \in D^i$ (whose score is P^i) **do**
    Give the reward $\tilde{R}(\tilde{s}, \tilde{s}')$ as defined in Equation 9.8
  end for
end for
Define $\hat{R}$:
for all transition $(\tilde{s}, \tilde{s}') \in D^i$ **do**
  $\hat{R}(\tilde{s}, \tilde{s}') = \tilde{R}(\tilde{s}, \tilde{s}')$
end for
for all $(\tilde{s}, a)$ in $\tilde{S} \times A$ **do**
  Update the Q-value $Q^{\pi_0}(\tilde{s}, a)$
end for
Update the policy: $\forall\ \tilde{s}, \pi_1(\tilde{s}) = \text{argmax}_a\, Q^{\pi_0}(\tilde{s}, a)$

repeat
  At iteration k,
  for all $\tilde{s}$ **do**
    Update the $\tilde{V}$-value $\tilde{V}^{\pi_k}(\tilde{s})$
  end for
  for all transition $(s, s') \in D^i$ **do**
    $\hat{R}(\tilde{s}, \tilde{s}') = \gamma F(\tilde{s}') - F(\tilde{s})$
    $F(\tilde{s}) = \tilde{V}^{\pi_k}(\tilde{s})$
    if the state $\tilde{s}'$ is the dialogue's final state $\tilde{s}_{t_f^i}$ **then**
      $F(\tilde{s}') = \hat{P}^i / \gamma^{t_f^i}$
    else
      $F(\tilde{s}') = \tilde{V}^{\pi_k}(\tilde{s}')$
    end if
  end for
  Update the policy: $\forall\ \tilde{s}, \pi_{k+1}(\tilde{s}) = \text{argmax}_a\, Q^{\pi_k}(\tilde{s}, a)$
until $\|\tilde{V}^{\pi_k} - \tilde{V}^{\pi_{k-1}}\| \leq \epsilon$

return R

Function Id	$F(\tilde{s_{t_f}})$	$F(\tilde{s_t})$, $\tilde{s_t}$ non-terminal
$\hat{R}^a$	$\gamma P/\gamma^{t_f}$	$E_{\pi^*}[P/\gamma^t \mid s_t]$
$\hat{R}^b$	$\gamma \hat{P}/\gamma^{t_f}$	$E_{\pi^*}[P/\gamma^t \mid s_t]$
$\hat{R}^c$	$\gamma \hat{P}/\gamma^{t_f}$	$E_{\pi^*}[\hat{P}/\gamma^t \mid s_t]$
$\hat{R}^d$	γP	$E_{\pi^*}[\gamma^{t_f-t} P \mid s_t]$
$\hat{R}^e$	$\gamma \hat{P}$	$E_{\pi^*}[\gamma^{t_f-t} P \mid s_t]$
$\hat{R}^f$	$\gamma \hat{P}$	$E_{\pi^*}[\gamma^{t_f-t} \hat{P} \mid s_t]$

Table 9.5: Parameters of the potential-based reward functions compared on the appointment scheduling simulator

and SVOR was trained on this corpus. Therefore, P is the estimation given by SVOR trained on the DINASTI corpus plus system task completion, and $\hat{P}$ is the estimation given by SVOR trained on the noisy DINASTI corpus plus system task completion. The estimation $\hat{P}$ is a noisy estimation of P.

The grid-based representation of the state space was once again built by performing entropy-based feature discretisation [Fayyad and Irani, 1993] on system task completion, number of dialogue turns, alternative system task completion, average ASR score and number of ASR rejections. The results of learning are displayed on Figure 9.4.

Figure 9.4 shows that R_a, R_b, and R_c consistently outperform all the other reward functions. Learning with R_d, R_e, and R_f is slightly faster than with R_{PS}, which accords with the reward shaping theory. It can be concluded here that learning with the interpretation of the performance scores as the return for the entire dialogue is faster than with the concurrent interpretation. Besides, learning seems to be robust to noise and there is no noticeable difference between R_b and R_c or between R_e and R_f. The function chosen for the rest of this chapter is R_b.

This chapter has shown that the interpretation of the performance scores proposed in the previous chapter leads to faster learning. This application of the reward shaping theory is promising because it is efficient on small corpora (500 dialogues) and it is relatively robust to noise. Next chapter proposes a second algorithm which differs from reward shaping in that it does not require an estimator such as SVOR and, from the trajectories, it directly estimates the values of the transitions, without relying on policy evaluation. This algorithm computes a reward function that minimises the Euclidean distance between the performance scores and the returns.

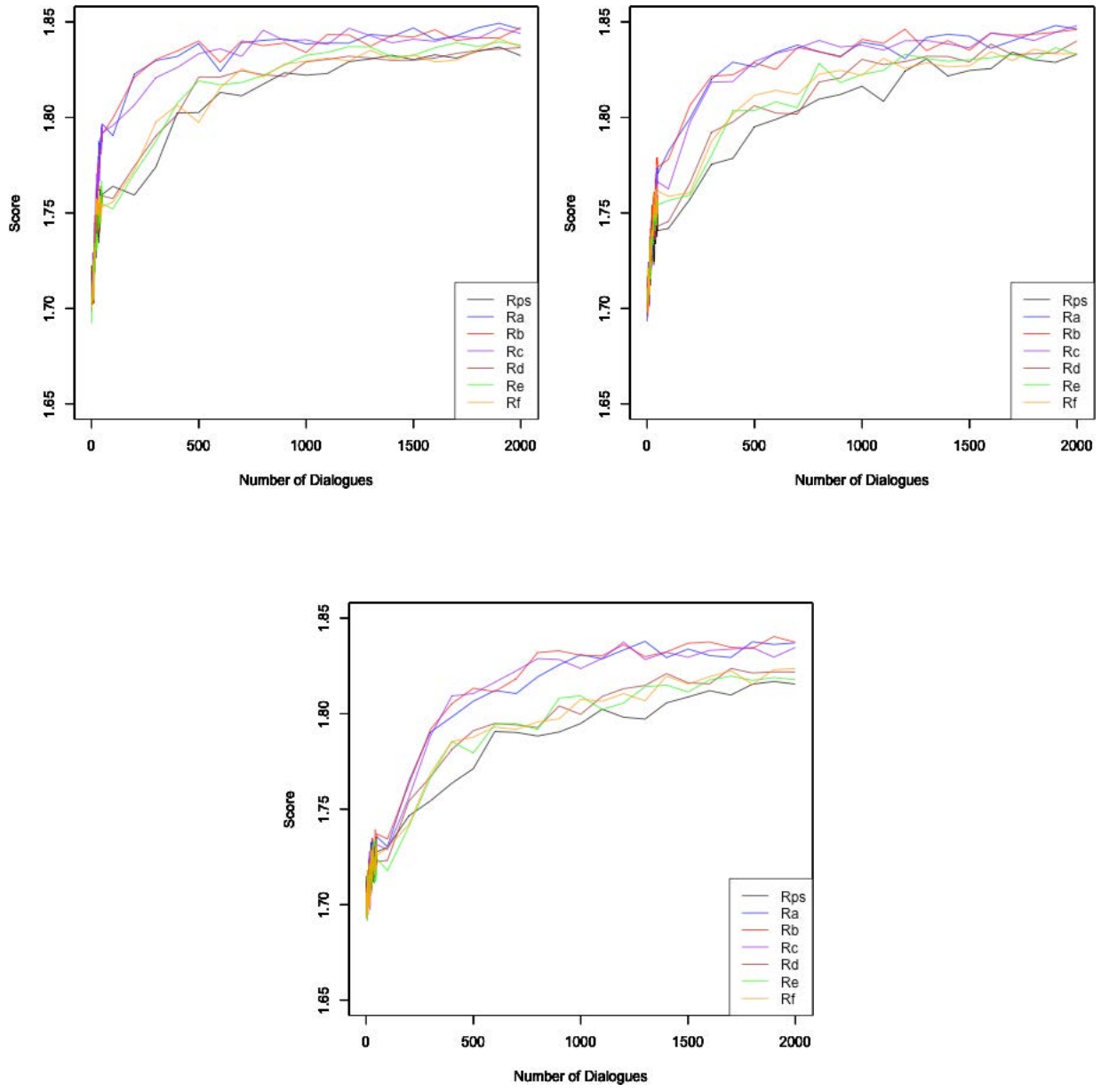


Figure 9.4: Comparison of several versions of reward shaping, from top left to bottom left, on corpora of 500, 1000 and 2000 dialogues (50 runs).

Chapter 10

Distance Minimisation

10.1 Formalism

Inverse Reinforcement Learning with Evaluation (IRLE) is another formulation of the inverse reinforcement learning problem where, instead of having examples of trajectories under an expert behaviour, it is supposed that an oracle which can decide the best of two policies is available Freire da Silva et al. [2006]. The inference problem presented in this paper is close to the one described by Freire da Silva et al.. Indeed, a utility function for the system is deduced from the evaluation of N dialogues $D_1, ..., D_N$ with performance scores $P_1, ..., P_N$. The difference here compared to the approach of Freire da Silva et al. is that an optimal policy cannot be inferred as the one preferred by the evaluator, only the reward function which best fits this evaluation can be found. Besides, instead of having relative evaluations of pairs of trajectories, the evaluator provides a numerical performance score for each dialogue so, the reward function which is closest to the evaluation model can be directly computed. The distance minimisation problem is formalised as follows:

Definition 3 *Let $\phi = [\phi_i]_{i=1,...,m}$ be a vector of basis functions over the transition space corresponding to $\tilde{S}$ ($\forall\, i \in [1,m], \forall\, (\tilde{s},\tilde{s}'), \phi_i(\tilde{s},\tilde{s}') \in [0,1]$), $P = [P^i]_{i=1,...,N}$ be the vector of performance scores, and let d_P be a distance measure between P and the reward vector $R = w^T\phi$. The distance minimisation problem seeks w^* such that $w^* = \text{argmin}_w\, d_P(w)$.*

Distance minimisation is based on the same interpretation of the performance scores as the reward shaping algorithm but simply performs a least-squares regression to compute the reward function. For a given dialogue D, the return at time 0 $r_0(D)$ is defined as a linear combination of the basis functions $\phi_1, ...\phi_m$:

$$\begin{aligned} r_0(D) &= \sum_{t\geq 0} \gamma^t R_t = \sum_{\tilde{s}_t,\tilde{s}'_t} \gamma^t \sum_{i=1}^{m} w_i \phi_i(\tilde{s}_t,\tilde{s}'_t) \\ &= \sum_{i=1}^{m} w_i \sum_{\tilde{s}_t,\tilde{s}'_t} \gamma^t \phi_i(\tilde{s}_t,\tilde{s}'_t) = w^{\mathrm{T}}\Phi(D) \qquad (10.1) \\ \text{with } \Phi(D) &= (\Phi_1(D)...\Phi_m(D))^{\mathrm{T}} \text{ and } \Phi_i(D) = \sum_{\tilde{s}_t,\tilde{s}'_t} \gamma^t \phi_i(\tilde{s}_t,\tilde{s}'_t) \end{aligned}$$

In what follows, Euclidean distance minimisation is used, so the reward function being looked for is the one which is closest to the evaluation from a purely numerical point of view. Section 10.5 discusses the choice of

the distance measure. The optimisation problem is the following:

$$\text{minimise } d_P(w) = \sum_{l=1}^{N}(r_0(D^l) - P(D^l))^2 = \sum_{l=1}^{N}(w^{\mathrm{T}}\Phi(D^l) - P(D^l))^2 \tag{10.2}$$

In a matrix form:

$$\text{minimise } \frac{1}{2}w^{\mathrm{T}}[2\sum_{l=1}^{N}\Phi(D^l)\Phi^{\mathrm{T}}(D^l)]w - w^{\mathrm{T}}[2\sum_{l=1}^{N}P(D^l)\Phi(D^l)] = w^{\mathrm{T}}[\frac{1}{2}Mw - b]$$
$$\text{with } M = 2\sum_{l=1}^{N}\Phi^{\mathrm{T}}(D^l)\Phi(D^l) \text{ and } b = 2\sum_{l=1}^{N}P(D^l)\Phi(D^l) \tag{10.3}$$

10.2 Resolution

The matrix M is symmetric. When M is positive and definite, the optimisation problem described in equation 10.3 has a unique solution and is tantamount to solving the equation $Mw = b$.

M is positive. Let $x \in \mathbb{R}^m : x^{\mathrm{T}}Mx = \sum_{i=1}^{m} x_i \sum_{j=1}^{m} x_j m_{i,j} = \sum_{D^l} 2\left(\sum_{i=1}^{m} x_i \Phi_i(D^l)\right)^2 \geq 0$. Under certain conditions, M is definite. Let $x \in \mathbb{R}^m$, according to the precedent derivation: $x^{\mathrm{T}}Mx = 0 \Leftrightarrow \forall\ D^l, \sum_i x_i\Phi_i(D^l) = 0$. Let us define $L = \left(\sum_{t_j \in D^l} \gamma^{t_j}\phi_i(\tilde{s}_{t_j}, \tilde{s}'_{t_j})\right)_{l,i}$. L is a rectangular matrix of size: the number of dialogues (N) $\times$ the number of transitions (m). M is definite if and only if m dialogues can be selected such that, on this corpus, all the transitions have been observed at least once and one cannot find a pair of transitions which would be systematically correlated in time. Indeed, if this corpus can be found, let L' be the matrix formed with the lines of M corresponding to these m dialogues. According to what precedes: $\forall\ x \in \mathbb{R}^m,\ L'x = 0$. L' was chosen so that it would have a rank equal to m so its kernel is the empty set. Therefore, $L'x = 0$ implies that $x = 0$ and M is definite.

When M is definite, the equation $Mw = b$ admits a unique solution which can be computed using either a direct (Cholesky decomposition,...) or an iterative (conjugate gradient,...) method. When M is not definite, the solution to the minimisation problem described in equation 10.3 is not unique. The problem can be solved using Tikhonov regularisation and then, it consists of searching for w^* which minimises $\|Mw - b\|^2 + \|\delta w\|^2$. The parameter δ can be fixed with the L-curve method or cross-validation.

The return at time 0 $r_0(D)$ for a given dialogue D of performance P is an estimation of the dialogue's performance score since, according to Equation 10.2, the rewards are computed so that $r_0(D)$ is close to P in terms of quadratic distance. The relevance of the estimation returned by the distance minimisation algorithm depends on the quality of the state space representation and especially the final states.

The theoretical properties of this algorithm have been studied and the analysis realized by Dr Bilal Piot from CRIStAL lab (UMR 9189) at CNRS/Lille 1/ECL and Dr Mathieu Geist from UMI 2958 at CNRS-GeorgiaTech is reproduced in Appendix D. These properties as well as tests on Garnets [Archibald et al., 1995, Piot et al., 2013, 2014] and on a different appointment scheduling simulator have been submitted for publishing [El Asri et al., 2016b]. The Garnets experiments, provided by Dr Bilal Piot are also reproduced in Appendix D. The aim of these experiments is to measure the performance of distance minimisation in terms of how the Euclidean error decreases with the number of dialogues.

10.3 Comparison of Reward Shaping and Distance Minimisation on the LTI corpus

In this section, the two approaches proposed for reward learning are compared. The two algorithms compute a reward function on $\tilde{S}$. The immediate reward distributed by reward shaping is $\gamma\tilde{V}^{\pi_k}(\tilde{s}') - \tilde{V}^{\pi_k}(\tilde{s})$. At state $\tilde{s}$, the dialogue manager receives a reward equal to the difference between the values of the arrival and the departure states. The values of these states depend on the DM's current policy π_k. The reward function computed by the distance minimisation algorithm, on the other hand, does not account for the agent's policy. The algorithm does not compute the value w_i of a transition $(\tilde{s}_i, \tilde{s}'_i)$ based on the current policy, but based on all the performance scores of the dialogues where this transition occurred. In this sense, even though the two algorithms are based on the same interpretation of the performance scores, the immediate rewards are based on different frameworks. On the one hand, the reward shaping algorithm computes an optimal policy and base the rewards on the values of the states according to this policy. On the other hand, the distance minimisation algorithm distributes rewards based on the mean values of the states, no matter the policy.

First, the reward shaping algorithm is compared to distance minimisation in terms of learning speed on the LTI corpus. The corpus of dialogues with System 3 is then used to highlight the importance of the design of the final state space for DiM and of the estimator $\hat{P}$ for RS.

For the reward shaping algorithm, real data with LTI is unavailable so it is not possible to train SVOR as estimator. Instead, a simple mean of scores for each final state is used, the estimator of the performance score for a dialogue D^i ending at state $\tilde{s}_{t_f^i}$ is defined as follows:

$$\begin{aligned} F(\tilde{s}_{t_f^i}) &= \text{average}\left(\frac{P(\tilde{s}_{t_f^i})}{\gamma^{t_f^i}}\right) \\ \hat{P}^i &= \gamma^{t_f^i}\,\text{average}\left(\frac{P(\tilde{s}_{t_f^i})}{\gamma^{t_f^i}}\right), \end{aligned} \tag{10.4}$$

For the distance minimisation algorithm, there were as many transition features ϕ_i as transitions τ_i: ϕ_i was equal to 1 at τ_i and 0 at the rest of the transition space.

10.3.1 Experimental Protocol

In total, 2300 dialogues were simulated with LTI but, as said in Chapter 4, only 600 dialogues were used to learn R_{RS} and R_{DiM}. This choice is explained by the fact that it is difficult, in real-life experiments to obtain as many as 2300 dialogues.

The reward shaping function R_{RS} and the distance minimisation function R_{DiM} are compared on the LTI corpus to the function R_{PS}, which is the reward function described in Chapter 4 and recalled here:

$$R_{PS} : (\tilde{s}, \tilde{s}') \in \tilde{S} \times \tilde{S} \mapsto \begin{cases} 0 & \text{if } \tilde{s}' \text{ is not final,} \\ -3\ \#EmptySlots + 0.25\ \#CorrectSlots & \\ -0.75\ \#WrongSlots - 0.015\ \#Turns & \text{otherwise.} \end{cases} \tag{10.5}$$

The final reward given by this function plays the role of the dialogue's performance score for the simulated dialogues. The performance score depends on the number of correct and incorrect slots and it decreases monotonously with the number of dialogue turns. As said in Chapter 4, the state space in LTI is defined in

Learning on 600 dialogues	Performance	Turns	Empty	confirmed
R_{RS}	0.13 ± 0.09	11.6	0	3
R_{DiM}	0.062 ± 0.10	7.72	0	0
R_{PS}	0.007 ± 0.10	8.55	0	0.8
Learning on 2300 dialogues	Performance	Turns	Empty	confirmed
R_{RS}	0.13 ± 0.09	11.6	0	3
R_{DiM}	0.08 ± 0.10	7.28	0	0.23
R_{PS}	0.04 ± 0.11	7.95	0	1.22

Table 10.1: 95% confidence interval for the mean performance, mean number of dialogue turns and mean number of empty and confirmed slots on 200 dialogues simulated with the policies learnt with R_{RS}, R_{DiM} and R_{PS} after 600 and 2300 dialogues.

terms of unknown, known, and confirmed slots. Consequently, the estimator proposed in Equation 10.4, which accounts for the number of turns but also the final state of the slots should highly correlate with the performance scores.

The constraint on the estimator $\hat{P}$ is that it should allow to learn an optimal policy which would also be optimal if the rewards were based on P. To ensure this property, the vector $\hat{P} = (\hat{P}^1, ..., \hat{P}^N)$ should be close to $P = (P^1, ..., P^N)$ both in terms of values and behaviour, which means that dialogues should be ranked the same with $\hat{P}$ and P and that the gap between the values of two dialogues according to P and according to $\hat{P}$ should be close.

Spearman's rank correlation coefficient and Euclidean distance were used to measure the proximity between the returns at time 0 with R_{RS} and R_{DiM} and the performance scores. The 1700 dialogues remaining from the 2300 dialogues were used to measure this proximity. 600 dialogues were uniformly drawn from the corpus of 2300 dialogues and the two reward inference algorithms were applied to this set. This process was repeated 100 times. On these 100 runs, the mean rank correlation coefficient is equal to 0.81 for R_{RS} and 0.84 for R_{DiM} and the average Euclidean distance is equal to 0.34 for R_{RS} and 0.37 for R_{DiM}. This confirms that the description of the state space is thorough enough to compute a good estimation given Equation 10.5. In addition, the information-feature space is here equal to the state space described in Chapter 4. In this case, given the simple performance score in Equation 10.5, accounting for the current state of each slot in the state space is enough to represent efficiently the performance scores.

Policies were first learnt on the 600 dialogues with R_{RS}, R_{DiM} and R_{PS} using the batch RL algorithm LSPI ([Lagoudakis and Parr, 2003]) and then on the whole corpus of 2300 dialogues. After a policy was learnt with LSPI, 200 new dialogues were simulated with the system following this policy. These new dialogues were then evaluated given the performance scoring function in Equation 10.5.

10.3.2 Results

The performance scores are reported in Table 10.1. The first observation that can be made is that, the policies learnt on 600 dialogues perform significantly better than the policy learnt with R_{PS}. This also confirms the fact that giving non-null intermediate rewards during the dialogue enables to speed up learning. Learning with R_{RS} is particularly efficient. Indeed, with R_{RS}, LSPI learns an efficient policy after only 600 dialogues. Although the policy learnt with R_{RS} leads to longer dialogues, it has the best evaluation. This can be explained by the

fact that this policy has a better success at confirming slots than the other two. A great number of confirmed slots implies a limited risk of getting one value wrong and since filling and having the right value for each slot has a greater weight in the scoring function (see Equation 10.5) than having short dialogues, the policy learnt with R_{RS} achieves better performance than the ones learnt with R_{DiM} and R_{PS}.

R_{PS} gives the exact scores as rewards which makes it more accurate than R_{RS} and R_{DiM} but this accuracy is counterbalanced by the fact that the rewards are only given at the end of each dialogue. The policy learnt with R_{PS} is the least competitive because it more poorly balances the trade-off between the number of confirmed slots and dialogue length than the other two policies.

10.4 Illustration on the System 3 corpus

Reward shaping and distance minimisation are now applied to the corpus of 740 evaluated calls with System 3. This application serves as an illustration, in a real-life setting, of the importance of the estimator for reward shaping and the final state space for distance minimisation.

10.4.1 Information-feature state space

The information-feature state space was again computed from features that were shown to be common predictors of user satisfaction [Larsen, 2003]. This list is not exhaustive and many other dialogue features should be added [Walker et al., 2002, Schmitt et al., 2012]. However, the object of this experiment is to give an illustration of the algorithms and a simple state space is sufficient for this purpose.

The set of dialogue features kept for System 3 are the following: the number of dialogue turns (`#turns`), the number of Automatic Speech Recognition (ASR) rejections (`#ASR_rejections`), the number of user time outs (`#time_outs`), and the ASR Score for the current user interaction (`ASRS`). These features are added to the informational component corresponding to the current phase of the dialogue (see Figure 4.1 in Chapter 4). The estimator $\hat{P}$ for reward shaping is defined as for LTI, by Equation 10.4 and thus depends on the final state encountered during the dialogue.

The information-feature space is once again built with feature discretisation. The resulting information-feature space is composed of the first 9 states listed in Table 10.2. Preliminary tests on the information-feature state space described in Section 10.4.1 showed that, when a dialogue did not end with module 7 or 8, the two algorithms did not compute an appropriate reward. As shown previously, RS requires a good estimator such as SVOR and DiM requires an appropriate final state space in order to estimate the performance of the system for a given dialogue. The policy learnt by the DM depends on the values of the final transitions for distance minimisation. Therefore, final states should only be final and never be visited before a dialogue ends. Otherwise, the performance estimation given by the return at time 0 will be erroneous. In order to overcome this drawback, a *hang up* state was added to the information-feature state space.

For the distance minimisation algorithm, once again, there were as many transition features ϕ_i as transitions τ_i: ϕ_i was equal to 1 at τ_i and 0 at the rest of the transition space.

10.4.2 Results

All the transitions between the information-feature states were observed at least once but Tikhonov regularisation had to be applied for the distance minimisation algorithm because the matrix M was not invertible. Indeed, as shown on Figure 4.1, module 5 is always followed by module 3. So, transition $5 \mapsto 3$ always comes

Informational Component	Dialogue features
The system proposes a time slot for the first time	
The system proposes a time slot and it is not the first proposition	
The system asks the user to propose a date	
The system proposes the date it has understood the user said	
The system has not understood the user's preferences	
The system has not understood the user's answer to a proposition	
An appointment has been scheduled	
The system has failed to schedule an appointment with the user	$\texttt{\#turns} < 12$
The system has failed to schedule an appointment with the user	$\texttt{\#turns} \geq 12$
The user has hung up.	

Table 10.2: The information-feature states for System 3.

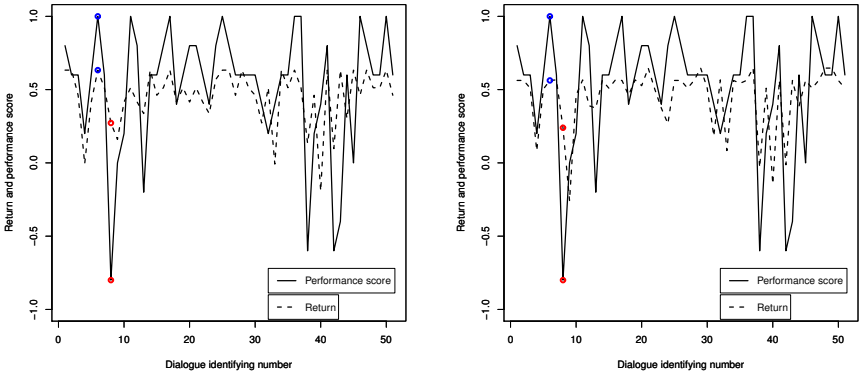
after transition $3 \mapsto 5$, which causes column dependencies in M. Other dependencies were observed which were not as direct as this one. The parameter of the regularisation was determined using the L-curve method.

The corpus was separated into a training set of 540 dialogues and a test set of 200 dialogues. The rank correlation coefficient is equal to 0.41 for R_{RS} and 0.39 for R_{DiM} and the average Euclidean distance is equal to 0.40 for R_{RS} and 0.39 for R_{DiM}. The rank correlation coefficient are much lower than the ones observed with the LTI corpus. This is explained by the fact that here, subjective user ratings are estimated, and not performance scores which depend on observable dialogue features as it was the case with LTI.

Figure 10.1 displays the returns computed by reward shaping and distance minimisation as well as the overall ratings given by the users. Globally, the returns inferred by both algorithms are coherent with the performance scores. However, Figure 10.1 shows that some dialogues are largely overrated by both inferred reward functions. For example, the eighth dialogue (red points in Figure 10.1) induced a positive return while user evaluation on this dialogue was highly negative. This phenomenon is due to the fact that Module 7 is not completely representative of task completion. Indeed, when users accepted an appointment which was not the one they had planned, it was considered that the task had not been achieved and users gave poor ratings to these kinds of dialogues. Therefore, task completion was not fully observable by the system and the reward functions tended to overrate dialogues that ended with module 7, which was, most of the time, synonymous with task completion and a high user rating. This also explains why dialogues ending with task completion seem underrated (blue points): the value of Module 7 (or the transitions to Module 7) included both successful dialogues and unsatisfactorily booked appointments. This experiment shows the importance of the design of the final state space for DiM and of the estimator for RS. It is important that the final state space is designed with the target to build a precise estimator of performance because the system's learning depends on it and the return at time 0 should give to the SDS designer a precise estimation of the system's performance so that online SDS monitoring is possible.

10.5 Discussion

As said for System 3, a *hang up* state was added to the information-feature state space. If, just as it is the case for System 3, the dialogue is modelled in such a way that, when the user hangs up, the system has failed achieving the task, then the *hang up* state can be estimated with no ambiguity. Nevertheless, this might not



(a) (b)

Figure 10.1: Comparison of performance estimation with the return computed by reward shaping (a) and the one computed by distance minimisation (b) on 50 dialogues.

always be the case. Paek and Pieraccini [2008] take the example of an SDS dedicated to airline reservation. Task completion depends on the aim of the user, which might be to make a reservation or just gather information about prices. If the users hang up, the system cannot know if it is because the users have had enough information or because they give up trying to get it. This reinforces the advocacy for a fine tuning of the final state space to the performance scores. It is possible to refine this final state space by adding dialogue features such dialogue length, ASR mean confidence score. Back to the airline reservation example, the *hang up* state, if associated with a large dialogue length and a small ASR mean score would probably enable to distinguish the dialogues that were ended by the user because they were problematic. On the other hand, if dialogue length is high but so is the mean ASR score, this might mean that the dialogue lasted long because the user kept asking questions to the system and then hung up once she had gathered all the information she wanted.

The distance minimisation algorithm performs a regression on the performance scores. Euclidean distance was chosen for this regression. One can argue that the Euclidean distance might not be the most appropriate choice if the aim is to imitate efficiently performance evaluation. For instance, although system learning will eventually be based on the numerical values of rewards, it might be preferable to infer a reward function which preserves the ranking of the dialogues established by the evaluation. In such a case, Spearman's rank correlation coefficient would be a more appropriate distance metric. This was the metric chosen by Sugiyama et al. [2012]. However, as shown in the experiments with LTI, even with high Spearman's rank correlation coefficient, learning was faster with reward shaping. Besides, the reward shaping algorithm theoretically ensures that the optimal policy with this reward function is equivalent to the one learnt with a reward function giving the decayed estimated performance score at the end of the dialogue and 0 for all the intermediate transitions.

10.6 Adaptation to GSDMRL

Following the methodology presented in the introductory section, RS and DiM are to be applied on a representation of the state space based on GSDMRL. For simplicity of presentation, these algorithms were described on a grid-based representation. Their adaptation to the GSDMRL representation is discussed in this section. Reward shaping on GSDMRL is described in Algorithm 5. Instead of having states $\tilde{s'}$ containing both an informational component and dialogue parameters, GSDMRL is based on a set of prototypes. The value function is defined on selection sets instead of states. As it was shown in Chapter 7, these selection sets have been re-arranged in order to represent homogeneous regions of the state space in terms of the Q-values. Each selection set can thus be seen as a grid state in that it represents a certain region of the state space. In a similar fashion, by projecting states onto selection sets, the distance minimisation algorithm can be easily adapted to GSDMRL. The only difference in this case concerns the value of the basis function $\phi_i(p, p')$ corresponding to a transition from s to s': it is equal to 1 if p is in the activation set of s and p' is in the activation set of s'.

This chapter has brought the last piece to the methodology introduced in the beginning of this thesis, that is to say, reward inference. Two algorithms were proposed, theoretically discussed and compared. It was shown that learning with these two functions was faster than by using the performance score as final reward. The following chapter will conclude this thesis and discuss possible future work.

Algorithm 5 Reward shaping algorithm on GSDMRL

Inputs:
The evaluated dialogues $D^1, ..., D^N$ with performance scores $P^1, ..., P^N$; an MDP$\backslash R$ $M\backslash R$; a genetic sparse distributed memory Mem, a stopping criterion ϵ
Define:
A starting random policy π_0 on each prototype p, $\forall\ (\tilde{s}, a), Q^{\pi_0}(p, a) = 0$.

Initialisation:
for all $D^i \in D^1, .., D^N$ **do**
  for all transition $(s, s') \in D^i$ (whose score is P^i) **do**
    Give the reward $\tilde{R}(s, s')$ as defined in Equation 9.8
  end for
end for
Define $\hat{R}$:
for all transition $(s, s') \in D^i$ **do**
  $\hat{R}(s, s') = \tilde{R}(s, s')$
end for
for all prototype-action pair (p, a) in Mem $\times$ A **do**
  Update the Q-value $Q^{\pi_0}(p, a)$
end for
Update the set-policies: $\forall\ p, \pi_1(p) = argmax_a Q^{\pi_0}(AS(p), a)$

repeat
  At iteration k,
  for all prototype p **do**
    Update the $\tilde{V}$-value $\tilde{V}^{\pi_k}(p)$
  end for
  for all transition $(s, s') \in D^i$ **do**
    Compute X the selection set of s and X' the selection set of s'
    $\hat{R}(s, s') = \gamma F(s') - F(s)$
    $F(s) = \tilde{V}^{\pi_k}(X) = \sum_{j=1}^{|X|} \frac{w^{j,i}}{\sum_{k=1}^{|X|} w^{k,i}} \tilde{V}(p^j)$
    if the state s' is the dialogue's final state $s_{t_f^i}$ **then**
      $F(s') = \hat{P}^i / \gamma^{t_f^i}$
    else
      $F(s') = \tilde{V}^{\pi_k}(X') = \sum_{j=1}^{|X'|} \frac{w^{j,i}}{\sum_{k=1}^{|X'|} w^{k,i}} \tilde{V}(p^j)$
    end if
  end for
  Update the set-policies: $\forall\ p, \pi_{k+1}(p) = argmax_a Q^{\pi_k}(AS(p), a)$
until $\|\tilde{V}^{\pi_k} - \tilde{V}^{\pi_{k-1}}\| \leq \epsilon$

return R

Chapter 11

Conclusion

11.1 Contributions

The work presented in this thesis aims at making reinforcement learning more practical for industry purposes. Indeed, although Spoken Dialogue Systems (SDS) have been widely spreading for the last few years, many of the deployed systems are limited in terms of functionality, robustness, and adaptivity. Reinforcement learning allows to overcome these limits. This thesis proposes a new methodology to easily plug reinforcement learning into a spoken dialogue system. The methodology is meant to be applied before system deployment for commercial use. The system's reinforcement learning parameters are defined according to data and a first policy is learnt. Then, the system can be directly deployed. This saves hours of fine-tuning parameters and it also relieves from the risk of deploying a system with a hand-crafted policy which would not meet real-usage requirements.

The methodology was designed to support the following claim: **from a dataset of N dialogues collected with varied dialogue management policies, automatically annotated with features on a dialogue turn-level basis, and evaluated by an expert or a user, it is possible to learn a dialogue management policy, without having to specify a state space representation nor a reward function**.

Steps of the methodology are recalled in Figure 11.1. To begin with, the spoken dialogue system developer designs a system with a state space composed of informational components and with a set of actions that the

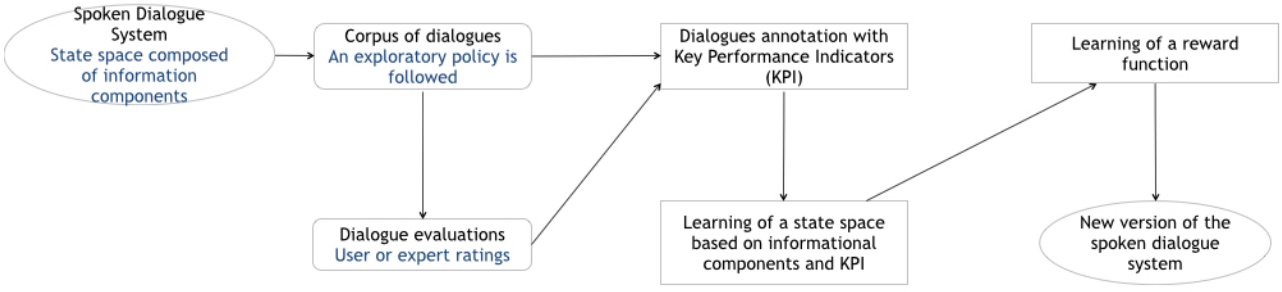


Figure 11.1: Methodology to automatically learn a policy from a set of rated dialogues.

system should learn how to choose. Then, this system is used for data collection with an exploratory policy. Afterwards, the dialogues in the corpus are annotated with a criterion to optimise such as user satisfaction and/or task completion, and with parameters such as dialogue duration, average speech recognition confidence score, etc. This annotated and rated corpus is used to learn, firstly, a representation of the state space and secondly, a reward function.

For the state space representation, this thesis introduces Genetic Sparse Distributed Memory for Reinforcement Learning (GSDMRL) [El Asri et al., 2016a], a new solution relying on the combination of a sparse distributed memory with genetic search. It is shown to be memory-efficient and to scale well with both the number of dialogues and number of features. Besides, this algorithm only requires to set two parameters which makes it easy to use by a non machine-learning expert developer. Finally, this representation is easily interpretable by the designer who can monitor her system's learning.

For the reward function inference, two algorithms are proposed: Reward Shaping and Distance Minimisation [El Asri et al., 2012, 2013, 2014c, 2016b]. These algorithms speed up learning compared to using directly the criterion to be optimised (such as user satisfaction). These algorithms are also easy to use and each only need one parameter to set. The methodology includes good practices to set efficiently the three aforementioned parameters.

These algorithms are validated on several problems ranging from reinforcement learning tasks such as mountain car to the dialogue tasks of appointment scheduling and restaurant-seeking.

11.2 Future perspectives

A first perspective is about the continuation of the methodology after the system has been deployed, i.e., evaluation and annotation of dialogues in order to refine the state space representation and reward function. Once it has been deployed, the system collects a lot of raw data, more than can be evaluated. Since a lot of dialogues are similar, or straightforward, an active selection of the dialogues inducing the highest amount of information is possible. To our knowledge, Active Learning [Settles, 2009] has been widely used for Natural Language Processing [Thompson et al., 1999, Bouneffouf et al., 2014], but never for dialogue systems, nor Inverse Reinforcement Learning [Russell, 1998].

Another interesting perspective of future work concerns Transfer Learning [Taylor and Stone, 2009, Lazaric, 2012] from a system to another. Indeed, if a company designs a system and learns a policy with this system, it would be interesting to transfer the knowledge acquired with this system to another one. This could save, for instance, from having to learn some generic actions such as confirmation demands for each newly designed system. Therefore, future work could consist in abstracting part of the state space, reward function, and policy learnt for a given system, and re-inject this knowledge into another system. Some similar work has been done for SDS for domain extension [Gašić et al., 2013] and user adaptation [Casanueva et al., 2015].

Finally, a third perspective is to adapt this work to incremental spoken dialogue systems [Schlangen and Skantze, 2011] which have been recently investigated in conjunction with reinforcement learning [Hatim Khouzaimi and Lefèvre, 2015]. Incremental dialogue strategies allow to have more human-like strategies by structuring the dialogue upon smaller units of time than a dialogue turn. The work proposed in this thesis can be adapted to this measure of time and it would be interesting to explore this area in future work.

Bibliography

Pieter Abbeel and Andrew Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proc. of ICML*, pages 1–8, 2004.

Kate Acomb, Jonathan Bloom, Krishna Dayanidhi, Peter Krogh, Esther Levin, and Roberto Pieraccini. Technical support dialog systems: Issues, problems, and solutions. In *In Proc. of ACL*, 2007.

Gregory Aist, James Allen, Ellen Campana, Carlos Gomez Gallo, Scott Stoness, Mary Swift, and Michael K. Tanenhaus. Incremental understanding in human-computer dialogue and experimental evidence for advantages over nonincremental methods. In *Proc. of Decalog*, 2007.

James S. Albus. A theory of cerebellar function. *Mathematical Biosciences*, 1971.

Ashraf Anwar, Dipankar Dasgupta, and Stan Franklin. Using genetic algorithms for sparse distributed memory initializations. In *Proc. of GECCO*, 1999.

Thomasarchi Archibald, Kenneth McKinnon, and Len Thomas. On the generation of Markov decision processes. *Journal of the Operational Research Society*, 1995.

J. L. Austin. *How to do Things with Words*. Clarendon Press, Oxford, 1962.

Merwan Barlier, Julien Perolat, Romain Laroche, and Olivier Pietquin. Human-machine dialogue as a stochastic game. In *Proc. of SIGDIAL*, 2015.

A. Barto and S. Mahadevan. *Recent Advances in Hierarchical Reinforcement Learning*. 2003.

Michael Baumann and Hans K. Buning. State aggregation by growing neural gas for reinforcement learning in continuous state spaces. In *Proc. of ICMLA*, 2011.

Michael Baumann, Timo Klerx, and Hans K. Büning. Improved state aggregation with growing neural gas in multidimensional state spaces. In *Proc. of ERLARS*, 2012.

R. E. Bellman. *Dynamic programming*. Princeton University Press, 1957.

Niels Ole Bernsen, Hans Dybkjær, and Laila Dybkjær. Principles for the design of cooperative spoken human-machine dialogue. In *Proc. of ICSLP*, pages 729–732, 1996.

Andrey Bernstein and Nahum Shimkin. Adaptive aggregation for reinforcement learning with efficient exploration: Deterministic domains. In *Proc. of COLT*, 2008.

Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.

Alan W Black, Susanne Burger, Alistair Conkie, Helen Hastie, Simon Keizer, Nicolas Merigaud, Gabriel Parent, Gabriel Schubiner, Blaise Thomson, D. Williams, Kai Yu, Steve Young, and Maxine Eskenazi. Spoken dialog challenge 2010: Comparison of live and control test results. In *Proc. of SIGDIAL*, 2010.

Dan Bohus and Alexander I. Rudnicky. Error handling in the RavenClaw dialog management architecture. In *Proc. of HLT-EMNLP*, 2005.

Johan Bos, Ewan Klein, Oliver Lemon, and Tetsushi Oka. DIPPER: Description and Formalisation of an Information-State Update Dialogue System Architecture. In *Proc. of SIGdial Workshop on Discourse and Dialogue*, 2003.

Abdeslam Boularias, Hamid R. Chinaei, and Brahim Chaib-draa. Learning the reward model of dialogue pomdps from data. In *Proc. of NIPS*, 2010a.

Abdeslam Boularias, Hamid R. Chinaei, and Brahim Chaib-draa. Learning the reward model of dialogue pomdps from data. In *Proc. of NIPS*, 2010b.

Djallel Bouneffouf, Romain Laroche, Tanguy Urvoy, Raphael Féraud, and Robin Allesiardo. Contextual bandit for active learning: Active thompson sampling. In *Proc. of ICONIP*, 2014.

Ronen I. Brafman and Moshe Tennenholtz. R-max - a General Polynomial Time Algorithm for Near-optimal Reinforcement Learning. *Machine Learning Research*, 2003.

M. Bratman. *Intentions, Plans, and Practical Reason*. 1987.

M. Bratman, D. Israel, and M. Pollack. Plans and resourcebounded practical reasoning. *Philosophy and AI: Essays at the Interface*, 1988.

P. Bretier. La communication orale coopérative : Contribution à la modélisation logique et à la mise en ?uvre d?un agent rationnel dialoguant, 1995.

Philippe Bretier, Romain Laroche, and Ghislain Putois. D5.3.4: Industrial self-help system ("system 3") adapted to final architecture. Technical report, CLASSIC Project, 2010.

D.S. Broomhead and David Lowe. Multivariable functional interpolation and adaptive networks. *Complex Systems*, 1988.

Inigo Casanueva, Thomas Hain, Heidi Christensen, Ricard Marxer, and Phil Green. Knowledge transfer between speakers for personalised dialogue management. In *Proc. of SIGDIAL*, 2015.

S. Chandramohan, M. Geist, F. Lefevre, and O. Pietquin. Clustering behaviors of spoken dialogue systems users. In *Proc. of ICASSP*, 2012.

Senthilkumar Chandramohan, Matthieu Geist, and Olivier Pietquin. Optimizing Spoken Dialogue Management with Fitted Value Iteration. In *Proc. of Interspeech*, 2010a.

Senthilkumar Chandramohan, Matthieu Geist, and Olivier Pietquin. Sparse Approximate Dynamic Programming for Dialog Management. In *Proc. of SIGDIAL*, 2010b.

Senthilkumar Chandramohan, Matthieu Geist, Fabrice Lefèvre, and Olivier Pietquin. User simulation in dialogue systems using inverse reinforcement learning. In *Proc. of Interspeech*, 2011.

Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2, 2011.

G. Chowdhary, Miao Liu, R. Grande, T. Walsh, J. How, and L. Carin. Off-policy reinforcement learning with gaussian processes. *IEEE/CAA Journal of Automatica Sinica*, 2014.

Wei Chu and S. Sathiya Keerthi. Support vector ordinal regression. *Neural Computation*, 19, 2007.

Jennifer Chu-Carroll. Mimic: An adaptive mixed initiative spoken dialogue system for information queries. In *Proc. of ANLP*, pages 97–104, 2000.

Jacob Cohen. A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20:37–46, 1960.

P. R. Cohen and H. J. Levesque. Persistence, intention, and commitment. *Intentions in Communication*, 1990.

Heriberto Cuayáhuitl, Steve Renals, Oliver Lemon, and Hiroshi Shimodaira. Hierarchical dialogue optimization using semi-markov decision processes. In *Proc. of EuroSpeech*, 2007.

Morena Danieli and Elisabetta Gerbino. Metrics for evaluating dialogue strategies in a spoken language system. In *Proc. of AAAI Spring Symposium on Empirical Methods in Discourse Interpretation and Generation*, 1995.

Lucie Daubigney, Matthieu Geist, Senthilkumar Chandramohan, and Olivier Pietquin. A Comprehensive Reinforcement Learning Framework for Dialogue Management Optimisation. *IEEE Journal of Selected Topics in Signal Processing*, 6(8):891–902, 2012.

Lucie Daubigney, Matthieu Geist, and Olivier Pietquin. Model-free POMDP optimisation of tutoring systems with echo-state networks. In *In Proc. of SIGDIAL*, 2013.

Harris Drucker, Chris J.C. Burges, Linda Kaufman, Alex Smola, and Vladimir Vapnik. Support Vector Regression Machines. In *Proc. of NIPS*, 1996.

Laila Dybkjaer, Niels Ole Bernsen, and Hans Dybkjaer. Grice incorporated: Cooperativity in spoken dialogue. In *Proc. of COLING*, 1996.

Laila Dybkjaer, Niels O. Bernsen, and Wolfgang Minker. Evaluation and usability of multimodal spoken language dialogue systems. *Speech Communication*, 43:33–54, 2004.

Jens Edlund, Joakim Gustafson, Mattias Heldner, and Anna Hjalmarsson. Towards human-like spoken dialogue systems. *Speech Communication*, 50, 2008.

Layla El Asri, Romain Laroche, and Olivier Pietquin. Reward function learning for dialogue management. In *Proc. of STAIRS*, 2012.

Layla El Asri, Romain Laroche, and Olivier Pietquin. Reward shaping for statistical optimisation of dialogue management. In *Proc. of SLSP*, 2013.

Layla El Asri, Hatim Khouzaimi, Romain Laroche, and Olivier Pietquin. Ordinal Regression for Interaction Quality Prediction. In *Proc. of ICASSP*, 2014a.

Layla El Asri, Romain Laroche, and Olivier Pietquin. DINASTI: Dialogues with a Negotiating Appointment Setting Interface. In *Proc. of LREC*, 2014b.

Layla El Asri, Romain Laroche, and Olivier Pietquin. Task completion transfer learning for reward inference. In *Proc. of MLIS*, 2014c.

Layla El Asri, Rémi Lemonnier, Romain Laroche, Olivier Pietquin, and Hatim Khouzaimi. NASTIA: Negotiating Appointment Setting Interface. In *Proc. of LREC*, 2014d.

Layla El Asri, Romain Laroche, and Olivier Pietquin. Compact and interpretable dialogue state representation with genetic sparse distributed memory. In *Proc. of IWSDS*, 2016a.

Layla El Asri, Bilal Piot, Mathieu Geist, Romain Laroche, and Olivier Pietquin. Score-based inverse reinforcement learning. In *Proc. of AAMAS*, 2016b.

Yaakov Engel, Shie Mannor, and Ron Meir. Reinforcement learning with gaussian processes. In *Proc. of ICML*, pages 201–208, 2005.

Klaus-Peter Engelbrecht, Florian Gödde, Felix Hartard, Hamed Ketabdar, and Sebastian Möller. Modeling user satisfaction with hidden markov model. In *Proc. SIGDIAL*, 2009.

Keelan Evanini, Phillip Hunter, Jackson Liscombe, David Suendermann, Krishna Dayanidhi, and Roberto Pieraccini. Caller experience: A method for evaluating dialog systems and its automatic prediction. In *Proc. of IEEE SLT*, pages 129–132, 2008.

Usama M. Fayyad and Keki B. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In *Proc. of UAI*, pages 1022–1027, 1993.

Jeffrey R.N. Forbes. *Reinforcement Learning for Autonomous Vehicles*. PhD thesis, University of California at Berkeley, 2002.

Valdinei Freire da Silva, Anna Helena Reali Costa, and Pedro Lima. Inverse reinforcement learning with evaluation. In *Proc. of ICRA*, 2006.

Jerome H. Friedman. Multivariate Adaptive Regression Splines. *The Annals of Statistics*, 19(1):1–67, 1991.

Michel Galley, Eric Fosler-lussier, and Ros Potamianos. Hybrid natural language generation for spoken dialogue systems. In *In Proceedings of the 7th European Conference on Speech Communication and Technology*, 2001.

M. Gašić, F. Jurčíček, S. Keizer, F. Mairesse, B. Thomson, K. Yu, and S. Young. Gaussian processes for fast policy optimisation of pomdp-based dialogue managers. In *Proc. of SIGDIAL*, 2010.

Milica Gašić, Matthew Henderson, Blaise Thomson, Pirros Tsiakoulis, and Steve Young. Policy optimisation of pomdp-based dialogue systems without state space compression. In *Proc. of SLT*, 2012.

Milica Gašić, Catherine Breslin, Matthew Henderson, Dongho Kim, Martin Szummer, Blaise Thomson, Pirros Tsiakoulis, and Steve Young. POMDP-based dialogue manager adaptation to extended domains. In *Proc. of SIGDIAL*, 2013.

Matthieu Geist and Olivier Pietquin. Algorithmic survey of parametric value function approximation. *IEEE Trans. Neural Netw. Learning Syst.*, 2013.

Kallirroi Georgila. Reinforcement learning of two-issue negotiation dialogue policies. In *Proc. of SIGDIAL*, 2013.

Kallirroi Georgila and David Traum. Reinforcement learning of argumentation dialogue policies in negotiation. In *Proc. of InterSpeech*, 2011.

Mohammad Ghavamzadeh, Alessandro Lazaric, Rémi Munos, and Matthew W. Hoffman. Finite-sample analysis of lasso-td. In *Proc. of ICML*, 2011.

J. Glass, J. Polifroni, S. Seneff, and V. Zue. Data collection and performance evaluation of spoken dialogue systems: The mit experience. In *Proc. of ICSLP*, 2000.

Geoffrey J. Gordon. Stable function approximation in dynamic programming. In *Proc. of ICML*, 1995.

A. Gorin, G. Riccardi, and J. H. Wright. How may i help you? *Speech Communications*, 23:113–127, 1997.

Paul Grice. *Studies in the Way of Words*, chapter Logic and Conversation. Harvard University Press, Cambridge MA, 1989.

Melita Hajdinjak and France Mihelic. A dialogue-management evaluation study. *Journal of Computing and Information Technology*, 2007.

A. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates, and A. Y. Ng. Deep speech: Scaling up end-to-end speech recognition. *ArXiv e-prints*, 2014.

Sunao Hara, Norihide Kitaoka, and Kazuya Takeda. Estimation method of user satisfaction using n-gram-based dialog history model for spoken dialog system. In *Proc. of LREC*, 2010.

Emma Hart and Peter Ross. Exploiting the analogy between immunology and sparse distributed memories: A system for clustering non-stationary data. In *Proc. of ICAIS*, 2002.

Mikko Hartikainen, EsaPekka Salonen, and Markku Turunen. Subjective evaluation of spoken dialogue systems using SERVQUAL method. In *Proc. of Interspeech*, pages 2273–2276, 2004.

Romain Laroche Hatim Khouzaimi and Fabrice Lefèvre. Optimising turn-taking strategies with reinforcement learning. In *Proc. of SIGDIAL*, 2015.

P.A. Heeman. Representing the reinforcement learning state in a negotiation dialogue. In *Proc. of ASRU*, 2009.

Tim A. Hely, David J. Willshaw, and Gillian M. Hayes. A new approach to kanerva's sparse distributed memory. *IEEE Transactions on Neural Networks*, 1997.

R. Higashinaka, K. Dohsaka, and H. Isozaki. Effects of self-disclosure and empathy in human-computer dialogue. In *In Proc. of SLT*, 2008.

Ryuichiro Higashinaka, Yasuhiro Minami, Kohji Dohsaka, and Toyomi Meguro. Issues in predicting user satisfaction transitions in dialogues: Individual differences, evaluation criteria, and prediction models. In *Spoken Dialogue Systems for Ambient Environments*, pages 48–60. 2010.

Geoffrey E. Hinton, Li Deng, Dong Yu, George E. Dahl, Abdel rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N. Sainath, and Brian Kingsbury. Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups. *IEEE Signal Process. Mag.*, 2012.

Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American statistical association*, 58, 1963.

Kate S. Hone and Robert Graham. Towards a tool for the subjective assessment of speech system interfaces (SASSI). *Natural Language Engineering*, 6:287–303, 2000.

Kate S. Hone and Robert Graham. Subjective assessment of speech-system interface usability. In *Proc. of Eurospeech*, pages 2082–2086, 2001.

Daniel Hsu, Sham M Kakade, and Tong Zhang. Random design analysis of ridge regression. *Foundations of Computational Mathematics*, 14, 2014.

Xuedong Huang, James Baker, and Raj Reddy. A historical perspective of speech recognition. *Communications of the ACM*, 2014.

Tobias Jung and Peter Stone. Feature selection for value function approximation using bayesian model selection. In *Proc. of ECML-PKDD*, 2009.

D. S. Jurafsky and J. H. Martin. *Speech and Language Processing*. Prentice Hall, 2000.

F. Jurčíček, S. Keizer, M. Gašić, F. Mairesse, B. Thomson, K. Yu, and S. Young. Real user evaluation of spoken dialogue systems using amazon mechanical turk. In *Proc. of Interspeech*, 2011.

Filip Jurčíček, Simon Keizer, Francois Mairesse, Kai Yu, Steve Young, Srinivanan Janarthanam, Helen Hastie, Xingkun Liu, and Oliver Lemon. D5.4: Proof-of-concept CLASSIC Appointment Scheduling system ("System 2"). Technical report, CLASSIC Project, 2010.

Leslie Pack Kaelbling. *Learning in Embedded Systems*. PhD thesis, 1990.

Pentti Kanerva. *Sparse Distributed Memory*. Cambridge, Mass.: Bradford/MIT Press., 1988.

Pentti Kanerva. *Associative Neural Memories: Theory and Implementation*. Oxford University Press, 1993.

Pentti Kanerva. Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors. *Cognitive computation*, 2009.

Hatim Khouzaimi, Romain Laroche, and Fabrice Lefèvre. An easy method to make dialogue systems incremental. In *Proc. of SIGDIAL*, pages 98–107, 2014a.

Hatim Khouzaimi, Romain Laroche, and Fabrice Lefèvre. Vers une approche simplifiée pour introduire le caractère incrémental dans les systèmes de dialogue. In *Proceedings of the TALN 2014 Conference*, 2014b.

Edouard Klein, Matthieu Geist, and Olivier Pietquin. Batch, off-policy and model-free apprenticeship learning. In *Proc. of IJCAI ALIHT*, 2011.

Edouard Klein, Matthieu Geist, Bilal Piot, and Olivier Pietquin. Inverse Reinforcement Learning through Structured Classification. In *Proc. of NIPS*, 2012.

K. Konolige and M. Pollack. A representationalist theory of intention. In *Proc. of IJCAI*, 1993.

Kostas Kostiadis and Huosheng Hu. KaBaGe-RL: Kanerva Based Generalisation and Reinforcement Learning for Possession Football. In *Proc. of IEEE IROS*, 2001.

Ronilda Lacson. The medical appointment scheduler. In *Proc. of MEDINFO*, 2004.

Michail G. Lagoudakis and Ronald Parr. Least-squares policy iteration. *Journal of Machine Learning Research*, 2003.

S. Lange and M. Riedmiller. Deep auto-encoder neural networks in reinforcement learning. In *Proc. of IJCNN*, 2010.

Sascha Lange, Thomas Gabel, and Martin Riedmiller. Batch reinforcement learning. In *Reinforcement Learning*, pages 45–73. 2012.

R. Laroche, B. Bouchon-Meunier, and P. Bretier. Uncertainty management in dialogue systems. In *Proc. of the European Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems*, 2008.

Romain Laroche. *Raisonnement sur les incertitudes et apprentissage pour les systèmes de dialogue conventionnels*. PhD thesis, Paris 6 University, 2010.

Romain Laroche, Ghislain Putois, Philippe Bretier, and Bernadette Bouchon-Meunier. Hybridisation of expertise and reinforcement learning in dialogue systems. In *Proc. of Interspeech*, 2009.

Romain Laroche, Philippe Bretier, and Ghislain Putois. Enhanced monitoring tools and online dialogue optimisation merged into a new spoken dialogue system design experience. In *Proc. of InterSpeech*, 2010a.

Romain Laroche, Ghislain Putois, and Philippe Bretier. Optimising a handcrafted dialogue system design. In *Proc. of Interspeech*, 2010b.

Romain Laroche, Ghislain Putois, and Philippe Bretier. Optimising a handcrafted dialogue system design. In *Proc. of InterSpeech*, 2010c.

Romain Laroche, Ghislain Putois, Philippe Bretier, Martin Aranguren, Julia Velkovska, Helen Hastie, Simon Keizer, Kai Yu, Filip Jurčíček, Oliver Lemon, and Steve Young. Report D6.4 : Final evaluation of classic towninfo and appointment scheduling systems. Technical report, CLASSIC Project, 2011.

Lars Bo Larsen. Issues in the evaluation of spoken dialogue systems using objective and subjective measures. In *Proc. of IEEE ASRU*, pages 209–214, 2003.

Staffan Larsson and David Traum. Information state and dialogue management in the trindi dialogue move engine toolkit. *Natural Language Engineering*, 6:323–340, 2000.

Alessandro Lazaric. Transfer in reinforcement learning: a framework and a survey. In *Reinforcement Learning*. 2012.

Oliver Lemon and Olivier Pietquin. Machine learning for spoken dialogue systems. In *Proc. of Interspeech*, pages 2685–2688, 2007.

Oliver Lemon and Olivier Pietquin. *Data-Driven Methods for Adaptive Spoken Dialogue Systems*. Springer, 2012.

Oliver Lemon, Kallirroi Georgila, James Henderson, and Matthew Stuttle. An ISU dialogue system exhibiting reinforcement learning of dialogue policies: Generic slot-filling in the TALK in-car system. In *Proc. of EACL*, 2006.

Esther Levin and Roberto Pieraccini. Concept-based spontaneous speech understanding system. In *In Proc. of Eurospeech*, 1995.

Esther Levin, Roberto Pieraccini, and Wieland Eckert. Learning dialogue strategies within the markov decision process framework. In *Proc. of IEEE ASRU*, 1997.

Baichuan Li, Zhaojun Yang, Yi Zhu, H. Meng, G. Levow, and I. King. Predicting user evaluations of spoken dialog systems using semi-supervised learning. In *Proc. of IEEE SLT*, pages 283–288, 2010.

Lihong Li, Jason D. Williams, and Suhrid Balakrishnan. Reinforcement learning for dialog management using least-squares policy iteration and fast feature selection. In *Proc. of Interspeech*, 2009.

Stephen Lin and Robert Wright. Evolutionary tile coding: An automated state abstraction algorithm for reinforcement learning. In *Proc. of AAAI Workshop on Abstraction, Reformulation, and Approximation*, 2010.

Diane Litman, Satinder Singh, Michael Kearns, and Marilyn Walker. Njfun: A reinforcement learning spoken dialogue system. In *In Proc. of the ANLP/NAACL Workshop on Conversational Systems*, 2000.

Diane J. Litman and Shimei Pan. Empirically evaluating an adaptable spoken dialogue system. In *Proc. of UMAP*, 1999.

Diane J. Litman and Shimei Pan. Designing and evaluating an adaptive spoken dialogue system. *User Modeling and User-Adapted Interaction*, 12:111–137, 2002.

Diane J. Litman and Scott Silliman. Itspoke: An intelligent tutoring spoken dialogue system. In *Demonstration Papers at HLT-NAACL*, 2004.

V. Louis. Conception et mise en ?uvre de modèles formels de calcul de plans d?action complexes par un agent rationnel dialoguant, 2002.

Klaus Macherey. *Statistical methods in natural language understanding and spoken dialogue systems*. PhD thesis, 2009.

Sridhar Mahadevan, Mauro Maggioni, and Carlos Guestrin. Proto-value functions: A laplacian framework for learning representation and control in markov decision processes. *Journal of Machine Learning Research*, 2006.

David L. Martin, Adam J. Cheyer, and Douglas B. Moran. The Open Agent Architecture: A Framework for Building Distributed Software Systems. *Applied Artificial Intelligence*, 1999.

Andrew K. McCallum. *Reinforcement Learning with Selective Perception and Hidden State*. PhD thesis, University of Rochester, 1995.

J. McCarthy. Ascribing mental qualities to machines. *Philosophical Perspectives in Artificial Intelligence*, 1979.

J. McCarthy and P. J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence*, 1969.

Toyomi Meguro, Ryuichiro Higashinaka, Kohji Dohsaka, Yasuhiro Minami, and Hideki Isozaki. Analysis of listening-oriented dialogue for building listening agents. In *In Proc. of SIGDIAL*, 2009.

Pascal Mercier. *Night Train to Lisbon*. 2004.

G. Meyer. Formalisation logique de préférences qualitatives pour la sélection de la réaction d'un agent rationnel dialoguant, 2006.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. In *Proc. of NIPS Deep Learning Workshop*, 2013.

Sebastian Möller. *Assessment and Prediction of Speech Quality in Telecommunications*. Kluwer, 2000.

Sebastian Möller. *Quality of telephone-based spoken dialogue systems*, chapter Quality of spoken dialogue systems. Springer, 2005.

Sebastian Möller, Paula Smeele, Heleen Boland, and Jan Krebber. Evaluating spoken dialogue systems according to de-facto standards: A case study. *Computer Speech Language*, 21(1):26–53, 2007.

Andrew W. Moore and Christopher G. Atkeson. The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces. *Machine Learning*, 1995.

A. Newell. The knowledge level. *AI Magazine*, 1980.

Andrew Y. Ng and Stuart Russell. Algorithms for inverse reinforcement learning. In *Proc. of ICML*, pages 663–670, 2000.

Andrew Y. Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proc. of ICML*, pages 278–287, 1999.

Tim Paek. Empirical methods for evaluating dialog systems. In *Proc. of ACL Workshop on Evaluation Methodologies for language and Dialogue Systems*, 2001.

Tim Paek. Reinforcement learning for spoken dialogue systems: Comparing strengths and weaknesses for practical deployment. In *Proc. of Interspeech, Dialog-on-Dialog Workshop*, 2006.

Tim Paek. Toward evaluation that leads to best practices: Reconciling dialog evaluation in research and industry. Technical report, Microsoft Research, 2007.

Tim Paek and David M. Chickering. The markov assumption in spoken dialogue management. In *Proc. of SIGdial Workshop on Discourse and Dialogue*, pages 35–44, 2005.

Tim Paek and Roberto Pieraccini. Automating spoken dialogue management design using machine learning : An industry perspective. *Speech Communication*, 50, 2008.

Maike Paetzel, Ramesh Manuvinakurike, and David DeVault. Knowledge acquisition strategies for goal-oriented dialog systems. In *Proc. of SIGDIAL*, 2015.

Nicolò Cesa-Bianchi Peter Auer and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47:235–256, 2002.

Roberto Pieraccini and Juan Huerta. Where do we go from here? research and commercial spoken dialog systems. In *Proc. of SIGDIAL*, 2005.

Roberto Pieraccini and Juan M. Huerta. *Where Do We Go from Here?* Springer Netherlands, 2008.

Olivier Pietquin. *A Framework for Unsupervised Learning of Dialogue Strategies*, pages 80–83. Presses universitaires de Louvain, 2004.

Olivier Pietquin. Machine Learning Methods for Spoken Dialogue Simulation and Optimization. In *Machine Learning*, pages 167–184. 2009.

Olivier Pietquin and Helen Hastie. Metrics for the evaluation of user simulations. Technical Report Deliverable 3.5, CLASSIC Project, 2010.

Olivier Pietquin and Steve Renals. ASR System Modeling For Automatic Evaluation And Optimization of Dialogue Systems. In *Proc. of ICASSP*, 2002.

Olivier Pietquin, Stéphane Rossignol, and Michel Ianotto. Training Bayesian networks for realistic man-machine spoken dialogue simulation. In *Proc. of IWSDS 2009*, 2009.

Bilal Piot, Matthieu Geist, and Olivier Pietquin. Learning from demonstrations: is it worth estimating a reward function? In *Proc. of ECML*, 2013.

Bilal Piot, Matthieu Geist, and Olivier Pietquin. Boosted and Reward-regularized Classification for Apprenticeship Learning. In *Proc. of AAMAS*, 2014.

Rajesh P. N. Rao and Olac Fuentes. Hierarchical learning of navigational behaviors in an autonomous robot using a predictive sparse distributed memory. *Machine Learning*, 1998.

Carl Edward Rasmussen and Malte Kuss. Gaussian processes in reinforcement learning. In *Proc. of NIPS*, 2004.

Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.

Bohdana Ratitch and Doina Precup. Sparse distributed memories for on-line value-based reinforcement learning. In *Proc. of ECML*, 2004.

Antoine Raux, Brian Langner, Allan Black, and Maxine Eskenazi. LET'S GO: Improving Spoken Dialog Systems for the Elderly and Non-natives. In *Proc. of Eurospeech*, 2003.

Verena Rieser and Oliver Lemon. Learning and evaluation of dialogue strategies for new applications: Empirical methods for optimization from small data sets. *Computational Linguistics*, 37, 2011.

David Rogers. Kanerva's sparse distributed memory: An associative memory algorithm well-suited to the connection machine. *International Journal of High Speed Computing*, 1988a.

David Rogers. Using data-tagging to improve the performance of the sparse distributed memory. Technical report, NASA, 1988b.

David Rogers. Statistical prediction with kanerva's sparse distributed memory. Technical report, NASA, 1989.

David Rogers. Weather prediction using a genetic memory. Technical report, NASA, 1990.

Antonio Roque, Hua Ai, and David Traum. Evaluation of an information state-based dialogue manager. In *Proc. of Brandial*, 2006.

Bertrand Russell. On denoting. 1905.

Stuart Russell. Learning agents for uncertain environments (extended abstract). In *Proc. of COLT*, 1998.

D. Sadek. Attitudes mentales et interaction rationnelle : vers une théorie formelle de la communicatio, 1991.

D. Sadek. Design considerations on dialogue systems : from theory to technology. the case of artimis. In *Proc. of IDS*, 1999.

D. Sadek, P. Bretier, and F. Panaget. Artimis : Natural dialogue meets rational agency. In *Proc. of IJCAI*, 1997.

Jean-Paul Sartre. *Situations*. 1990.

Jost Schatzmann, Karl Weilhammer, Matt Stuttle, and Steve Young. A survey of statistical user simulation techniques for reinforcement-learning of dialogue management strategies. *The Knowledge Engineering Review*, 21(2), 2006.

David Schlangen and Gabriel Skantze. A general, abstract model of incremental dialogue processing. *Dialogue and Discourse*, 2:83–111, 2011.

Alexander Schmitt, Carolin Hank, and Jackson Liscombe. Detecting problematic dialogs with automated agents. In *Perception in Multimodal Dialogue Systems*, Lecture Notes in Computer Science, pages 72–80. 2008.

Alexander Schmitt, Benjamin Schatz, and Wolfgang Minker. Modeling and predicting quality in spoken human-computer interaction. In *Proc. of SIGDIAL*, 2011.

Alexander Schmitt, Stefan Ultes, and Wolfgang Minker. A parameterized and annotated spoken dialog corpus of the cmu let's go bus information system. In *Proc. of LREC*, 2012.

Ethan Selfridge, Iker Arizmendi, Peter Heeman, and Jason D. Williams. Continuously predicting and processing barge-in during a live spoken dialogue task. In *Proc. of SIGDIAL*, 2013.

Burr Settles. Active learning literature survey. Computer Sciences Technical Report 1648, University of Wisconsin–Madison, 2009.

Satinder Singh and Richard S. Sutton. Reinforcement learning with replacing eligibility traces. In *Machine Learning*, 1996.

Satinder Singh, Michael Kearns, Diane Litman, and Marilyn Walker. Reinforcement learning for spoken dialogue systems. In *Proc. of NIPS*, 1999.

Satinder Singh, Michael Kearns, Diane Litman, and Marilyn Walker. Optimizing dialogue management with reinforcement learning: experiments with the njfun system. *Journal of Artificial Intelligence Research*, 16: 105–133, 2002.

Edward Jay Sondik. The optimal control of partially observable markov processes. Technical report, Stanford Electronics labs, 1971.

Alexander L. Strehl and Michael L. Littman. A theoretical analysis of model-based interval estimation. In *Proc. of ICML*, 2005.

Sofia Strombergsson, Anna Hjalmarsson, Jens Edlund, and David House. Timing responses to questions in dialogue. In *Proc. of Interspeech*, 2013.

J. Sturm, E. Den Os, and L Boves. Issues in spoken dialogue systems: Experiences with the dutch arise system. In *Proc. of ESCA Workshop on Interactive dialogue in multi-modal systems*, 1999.

Hiroaki Sugiyama, Toyomi Meguro, and Yasuhiro Minami. Preference-learning based Inverse Reinforcement Learning for Dialog Control. In *Proc. of Interspeech*, 2012.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning. An introduction*. MIT Press, 1998.

G. Taylor and R. Parr. Kernelized value function approximation for reinforcement learning. In *In Proc. of ICML*, 2009.

Matthew E. Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10:1633–1685, 2009.

Gerald Tesauro. Temporal difference learning and td-gammon. *Commun. ACM*, 38, 1995.

Cynthia A Thompson, Mary Elaine Califf, and Raymond J Mooney. Active learning for natural language parsing and information extraction. In *Proc. of ICML*, 1999.

Robert Tibshirani. Regression shrinkage and selection via the LASSO. *Journal of the Royal Statistical Society. Series B (Methodological)*, 1996.

Andrey Tikhonov. Solution of incorrectly formulated problems and the regularization method. In *Soviet Mathematics*, 1963.

S. Timmer and M. Riedmiller. Fitted Q Iteration with CMACs. In *Proc. of ADPRL*, 2007.

Dave Toney, Johanna Moore, and Oliver Lemon. Evolving optimal inspectable strategies for spoken dialogue systems. In *Proc. of NAACL-HLT*, 2006.

David Traum and Staffan Larsson. *The Information State Approach to Dialogue Management*. 2003.

J.N. Tsitsiklis and B. Van Roy. An analysis of temporal-difference learning with function approximation. *Automatic Control, IEEE Transactions on*, 1997.

Stefan Ultes and Wolfgang Minker. Improving interaction quality recognition using error correction. In *Proc. of SIGDIAL*, 2013.

Stefan Ultes, Alexander Schmitt, and Wolfgang Minker. Towards quality-adaptive spoken dialogue management. In *Proc. of NAACL-HLT SDCTD*, 2012.

Vladimir N. Vapnik. *Statistical Learning Theory*. 1998.

M. Walker, D. Hindle, J. Fromer, G.D. Fabbrizio, and C. Mestel. Evaluating competing agent strategies for a voice e-mail agent. In *Proc. of EuroSpeech*, 1997a.

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

Marilyn A. Walker, Diane J. Litman, Candace A. Kamm, and Alicia Abella. PARADISE: a framework for evaluating spoken dialogue agents. In *Proc. of EACL*, pages 271–280, 1997b.

Marilyn A. Walker, Jeanne C. Fromer, and Shrikanth Narayanan. Learning optimal dialogue strategies: A case study of a spoken dialogue agent for email. In *Proc. of COLING/ACL*, pages 1345–1352, 1998.

Marilyn A. Walker, Candace A. Kamm, and Diane J. Litman. Towards developing general models of usability with PARADISE. *Natural Language Engineering*, 1999.

Marilyn A. Walker, Irene Langkilde-Geary, Helen Wright Hastie, Jerry Wright, and Allen Gorin. Automatically training a problematic dialogue predictor for a spoken dialogue system. *Journal of Artificial Intelligence Research*, 16:293–319, 2002.

Christopher J. Watkins. *Learning with delayed rewards*. PhD thesis, Psychology Department, University of Cambridge, 1989.

Paul Weng. Ordinal Decision Models for Markov Decision Processes. In *Proc. of ECAI*, 2012.

Steven D. Whitehead and Dana H. Ballard. Learning to Perceive and Act by Trial and Error. *Machine Learning*, 7(1):45–83, 1991.

Jason D. Williams. The best of both worlds : Unifying conventional dialog systems and pomdps. In *Proc. of Interspeech*, 2008.

Jason D. Williams and Steve Young. Partially observable markov decision processes for spoken dialog systems. *Computer Speech and Language*, 21:231–422, 2007.

Silke Witt. A global experience metric for dialog management in spoken dialog systems. In *Proc. of SemDial*, pages 158–166, 2011.

Cheng Wu and Waleed Meleis. Adaptive fuzzy function approximation for multi-agent reinforcement learning. In *Proc. of IEEE/WIC/ACM IAT*, 2009.

Zhaojun Yang, Baichuan Li, Yi Zhu, I. King, G. Levow, and H. Meng. Collection of user judgments on spoken dialog system with crowdsourcing. In *Proc. of IEEE SLT*, pages 277–282, 2010.

Nicole Yankelovitch. How do users know what to say ? *Interactions*, 3:32–43, 1996.

Steve Young. Using POMDPs for dialog management. In *Proc. of SLT*, 2006.

Steve Young, Simon Keizer, Francois Mairesse, Jost Schatzmann, Blaise Thomson, and Kai Yu. The Hidden Information State model: A practical framework for POMDP-based spoken dialogue management. *Computer Speech and Language*, 2009.

Huizhen Yu and D.P. Bertsekas. Basis function adaptation methods for cost approximation in mdp. In *In Proc. of ADPRL*, 2009.

Brian D. Ziebart, Andrew Maas, J. Andrew Bagnell, and Anind K. Dey. Maximum entropy inverse reinforcement learning. In *Proc. of AAAI ICAI*, pages 1433–1438, 2008.

Appendix A

Features in the LEGO corpus

Feature name	Feature signification
ASRConfidence	ASR confidence score for the current exchange
MeanASRConfidence	Average ASR confidence score up to the current exchange
(Mean)ASRConfidence	Average ASR confidence score for the last three exchanges
#ASRRejections	Number of ASR rejections up to the current exchange
(#)ASRRejections	Number of ASR rejections during the last three exchanges
%ASRRejections	Percentage of ASR rejections relative to all previous exchanges
#TimeOuts	Number of user time outs up to the current exchange
(#)TimeOuts	Number of user time outs during the last three exchanges
%TimeOuts	Percentage of user time outs relative to all previous exchanges
WPUT	Mean number of words per user turn up to the current exchange
SDA	System Dialogue Act: SDA_GREETING, SDA_OFFER_HELP, SDA_ASK_BUS, SDA_ASK_CONFIRM_DEPARTURE, SDA_CONFIRM_UNDERSTOOD, SDA_ASK_DESTINATION, SDA_ASK_CONFIRM_DESTINATION, SDA_FILLER, SDA_ASK_TIME, SDA_ASK_CONFIRM_TIME, SDA_ANNOUNCE_QUERYING, SDA_DISAMBIGUATE_BUS_STOP, SDA_ASK_ANOTHER_QUERY, SDA_GOODBYE, SDA_ASK_CONFIRM_BUS, SDA_ASK_DEPARTURE, SDA_EXPLAIN, SDA_DELIVER_RESULT, SDA_INFORM_NO_ROUTE, SDA_ANNOUNCE_RESTART, SDA_INFORM_NO_SCHEDULE, SDA_ASK_CONFIRM_NEIGHBORHOOD, SDA_ASK_CONFIRM_WITH_KEYS, SDA_INFORM_SHORTER_ANSWER, SDA_INFORM_HELP, SDA_INSTRUCT_MORE_QUIET, SDA_INSTRUCT_LOUDER,
UDA	User Dialogue Act:UDA_LINE_INFORMATION, UDA_CONFIRM, UDA_CONFIRM_DEPARTURE, UDA_PLACE_INFORMATION, UDA_CONFIRM_DESTINATION, UDA_TIME_INFORMATION, UDA_CONFIRM_TIME, UDA_INFORM, UDA_NEW_QUERY, UDA_UNQUALIFIED_UNRECOGNIZED, UDA_GOODBYE, UDA_CONFIRM_BUS, UDA_REQUEST_SCHEDULE, UDA_REQUEST_PREVIOUS_BUS, UDA_REQUEST_NEXT_BUS, UDA_REJECT, UDA_REJECT_TIME, UDA_REJECT_BUS, UDA_REJECT_DESTINATION, UDA_REQUEST_HELP, UDA_POLITE
DD	Dialogue duration up to the current exchange
#SystemQuestions	Number of questions asked by the system up to the current exchange
(#)SystemQuestions	Number of questions asked by the system during the last three exchanges
WPST	Mean number of words per system turn up to the current exchange
#systemTurns	Number of system turns up to the current exchange
#userTurns	Number of user turns up to the current exchange
#RePrompts	Number of re-prompts up to the current exchange
(#)RePrompts	Number of re-prompts during the last three exchanges
%RePrompts	Percentage of re-prompts relative to all previous exchanges

Table A.1: Automatically Computable Features in the LEGO corpus.

Appendix B

User guide provided to the volunteers who interacted with NASTIA

B.1 DINASTI Experimentation: User Guide

This guide is addressed to the volunteers for the experiment with the appointment setting vocal interface designed in the context of the DINASTI project (DIalogues with a Negotiating Appointment SeTting Interface).

First of all, thanks for your participation!

B.1.1 Course of the experimentation

Context

Today is Monday, July 12th and your landline is not functional. After it diagnosed that the intervention of an engineer on site was required, the technical service has redirected you to a spoken dialogue system to book an appointment. Your aim is to set an appointment at one of the available slots on the following calendar.

You are invited to interact with the system five times. For each interaction, you will have a two-week calendar where the green cells will correspond to your available slots and the red cells will correspond to your unavailable slots. Your task is to set an appointment during one of your available slots.

After each interaction, you will be asked to fill in a short questionnaire to evaluate the quality of the interaction and report any difficulty encountered during the utilization of the service.

You are not asked to perform the five interactions in a row. For each dialogue, we advise you to settle in a calm environment in order to benefit from optimal speech recognition.

Interaction with the service You can interact with the system in natural language. If you face a difficulty or are not sure of what you can say, you can ask for help. The interface will then explain what you can say. During the dialogue, the following vocal commands are available:

- “Repeat”: the interface repeats its latest utterance.
- “Help”: the interface provides help on what you can say.

Figure B.1: Web interface for a dialogue with the appointment setting service.

The dialogue

Access to the vocal interface:
Click on one of the five links that were emailed to you. Each link contains an identifier that will enable the service to identify the scenario you will follow. Each link will get you to a web page similar to the one in Figure B.1. Two phone numbers are displayed beneath the calendar. If a number is busy, you can use the second one. For each call, you will be asked to dial the code corresponding to the link on which you have clicked (61887698 in the example above). You can now call the service and try to set an appointment!

Evaluation

After each dialogue, you are invited to click on "Evaluate this dialogue". An evaluation questionnaire will then be displayed on the screen. In the free text cell, please feel free to report any remark on the service or any difficulty with the service. An example of filled questionnaire is given in Figure B.2. After submitting the questionnaire, you will be able to proceed to another one of the five interactions on the list that was emailed to you. Thank you for your participation in this project!

Evaluation du dialogue code 61887698

1. **Avez-vous obtenu un rendez-vous ?**

   (•) Oui ( ) Non ( ) Je ne sais pas

2. **Ce rendez-vous correspondait-il à un emplacement libre dans votre agenda ?**

   (•) Oui ( ) Non ( ) Je ne sais pas

3. **Quand avez-vous pris rendez-vous ?**

   Date [Mercredi 21 Juillet] Heure [Après-midi]

4. **Lors de votre intéraction avec le service, vous saviez comment vous exprimer.**

   ( ) Tout à fait d'accord (•) D'accord ( ) Plutôt d'accord ( ) Plutôt pas d'accord ( ) Pas d'accord ( ) Pas du tout d'accord

5. **Vous pouviez aisément corriger les erreurs de compréhension du service.**

   ( ) Tout à fait d'accord ( ) D'accord (•) Plutôt d'accord ( ) Plutôt pas d'accord ( ) Pas d'accord ( ) Pas du tout d'accord

6. **Vous compreniez facilement ce que disait le service.**

   (•) Tout à fait d'accord ( ) D'accord ( ) Plutôt d'accord ( ) Plutôt pas d'accord ( ) Pas d'accord ( ) Pas du tout d'accord

7. **Le service vous a fourni suffisamment d'informations pour que le dialogue soit facile à suivre.**

   ( ) Tout à fait d'accord ( ) D'accord (•) Plutôt d'accord ( ) Plutôt pas d'accord ( ) Pas d'accord ( ) Pas du tout d'accord

8. **L'intéraction avec le service a été efficace.**

   ( ) Tout à fait d'accord (•) D'accord ( ) Plutôt d'accord ( ) Plutôt pas d'accord ( ) Pas d'accord ( ) Pas du tout d'accord

9. **L'intéraction avec le service a été fluide.**

   ( ) Tout à fait d'accord ( ) D'accord (•) Plutôt d'accord ( ) Plutôt pas d'accord ( ) Pas d'accord ( ) Pas du tout d'accord

10. **Le service était concis.**

    ( ) Tout à fait d'accord ( ) D'accord (•) Plutôt d'accord ( ) Plutôt pas d'accord ( ) Pas d'accord ( ) Pas du tout d'accord

11. **Quelle note attribueriez-vous à ce dialogue (note de 1 à 10, 10 étant la note la plus élevée) ?**

    [9]

12. **Avez-vous des remarques ou commentaires ?**

    Le service n'a pas compris "mercredi vingt-et-un".

[Soumettre]

Contact : Expérimentation DINASTI

Figure B.2: Example of filled evaluation questionnaire.

Appendix C

Features in the DINASTI corpus

Feature name	Feature signification
#DecisionTurns	Number of decisions up to this turn. A decision turn is a dialogue turn where the system has to choose between different possible actions.
#SystemTurns	Number of system turns up to this turn.
#UserTurns	Number of user turns up to this turn.
#RePrompts	Number of re-prompts up to this turn.
#ASRRejections	Number of speech recognition rejections up to this turn.
#TimeOuts	Number of user time outs up to this turn.
ASRConfidence	Mean ASR confidence score up to this turn.
#SystemQuestions	Number of system questions up to this turn.
#HelpMessages	Number of prompted help messages up to this turn.
WPST	Mean number of words per system turn up to this turn.
WPUT	Mean number of words per user turn up to this turn.
DD	Dialogue duration in seconds.
#SystemDialogueActs (18 features)	Number of times each system dialogue act has been performed. System dialogue acts are: SDA_GREETING, SDA_GOODBYE, SDA_INFORM, SDA_REPAIR, SDA_ASK_DAY, SDA_ASK_DATE, SDA_ASK_OTHER_PERIOD, SDA_NOT_AVAILABLE, SDA_ASK_CONFIRMATION, SDA_DATE_PROPOSITION, SDA_ASK_WHICH, SDA_ASK_PERIOD, SDA_LIST, SDA_ASK_WEEK SDA_ASK_OTHER_WEEK, SDA_REPEAT, SDA_ERROR
#UserDialogueActs (9 features)	Number of times each user dialogue act has been performed. User dialogue acts are: UDA_NOINPUT, UDA_NOMATCH, UDA_CONFIRM, UDA_PROPOSE_DATE, UDA_ASK_HELP, UDA_CONTRADICT UDA_DO_NOT_KNOW, UDA_ASK_REPEAT, UDA_SAY_NONE
#SystemNegociationStrategies (4 features)	Number of times each negotiation strategy has been chosen: SNS_LIST, SNS_SYS_INIT, SNS_USER_INIT, SNS_SYS_PROPOSITION
#RuleUsage (38 features)	Number of times each grammar rule has been triggered
#TagUsage (39 features)	Number of times each word tag has been recognised

Table C.1: Features in the DINASTI corpus.

Feature name	Feature signification
SDA_GREETING	Greet the user
SDA_GOODBYE	Say goodbye to the user
SDA_INFORM	Provide information about the system's calendar
SDA_REPAIR	Recover from ASR rejection or user time out
SDA_ASK_DAY	Ask the user on which day they are available
SDA_ASK_DATE	Ask the user when she/he is available
SDA_ASK_OTHER_PERIOD	Ask the user if she/he is available during the morning or the afternoon, after she/he has refused an appointment the same day respectively during the afternoon or the morning.
SDA_NOT_AVAILABLE	Inform the user that a slot in not available
SDA_ASK_CONFIRMATION	Ask the user for a confirmation
SDA_DATE_PROPOSITION	Propose a slot to the user
SDA_ASK_WHICH	After having proposed a list of 4 slots, ask the user if a slot is suitable
SDA_ASK_PERIOD	Ask the user if they are available during the morning or the afternoon
SDA_LIST	Propose a list of available slots
SDA_ASK_WEEK	Ask the user on which week they are available
SDA_ASK_OTHER_WEEK	Ask the user if they are available the other week
SDA_REPEAT	Repeat the last system prompt
SDA_ERROR	Inform the user an error has occurred

Table C.2: NASTIA's dialogue acts.

Feature name	Feature signification
UDA_NOINPUT	User time out
UDA_NOMATCH	ASR rejection
UDA_CONFIRM	The user confirms what the system has understood
UDA_PROPOSE_DATE	The user expresses constraints
UDA_ASK_HELP	The user requests the help section
UDA_CONTRADICT	The user contradicts what the system has understood
UDA_ASAP	The user says she/he wants to set an appointment as fast as possible
UDA_ASK_REPEAT	The user asks the system to repeat
UDA_SAY_NONE	The user answers “none” after the system has proposed four slots

Table C.3: User dialogue acts recognised by NASTIA.

Appendix D

Theoretical Properties of Distance Minimisation

The theoretical properties have been studied by Dr Bilal Piot from CRIStAL lab (UMR 9189) at CNRS/Lille 1/ECL and Dr Mathieu Geist from UMI 2958 at CNRS-GeorgiaTech. We reproduce this study here with their approval. The proofs of the theorems are not reproduced here but have been submitted for publication [El Asri et al., 2016b].

D.1 Formal description

A Markov Decision Process (MDP) is a tuple $\{\mathcal{S}, \mathcal{A}, P, \gamma, r\}$ where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the finite action space, $P(ds'|s,a)$ the Markovian transition kernel on $\mathcal{S}$, $\gamma \in (0,1)$ the discount factor and $r : \mathcal{S} \to \mathbb{R}$ the (bounded) reward function. A policy is a mapping $\pi : \mathcal{S} \to \mathcal{A}$. For a policy π, we define the transition kernel $P_\pi(ds'|s) = P(ds'|s,\pi(s))$. The quality of a policy is quantified by the related value function, that associates to each state the expected discounted sum of rewards received from starting in state s and following the policy π: $v^r_\pi(s) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r(S_t)|S_0 = s, S_{t+1} \sim P_\pi(.|S_t)]$. An optimal policy π^r_* (respectively to the reward r) is such that the related value function $v^r_{\pi^r_*} = v^r_*$ satisfies componentwise $v^r_* \geq v^r_\pi$, for any policy π.

In this work, the true reward r is unknown and has to be estimated. To do so, we adopt a linear parameterization of the reward and we assume the availability of a set of trajectories scored by a human expert. More formally, the reward is parameterized by a feature vector $\phi : \mathcal{S} \to \mathbb{R}^d$, $r_\theta(s) = \theta^\top \phi(s)$. The available data is a set $\mathcal{D} = \{(h_i, v_i)_{1 \leq i \leq n}\}$, where $h_i = (s^i_0, \ldots, s^i_{T_i}) = (s^i_j)_{j=0}^{T_i}$ is a trajectory of length $T_i + 1$ and v_i is the score (seen as the discounted sum of rewards of h_i) given by a human expert for this trajectory. Notice that we do not make any specific assumption about how the trajectory is generated (except that it is a trajectory, obtained by applying successively T_i actions on encountered states, next states being drawn according to the dynamics). Data are assumed independently and identically distributed (which does not mean that states of a given trajectory are independent, obviously). For a given trajectory $h = (s_t)_{t=0}^T$ and a given reward r_θ, the discounted sum of rewards can be written as

$$\sum_{t=0}^{T} \gamma^t r_\theta(s_t) = \theta^\top \mu(h) \text{ with } \mu(h) = \sum_{t=0}^{T} \gamma^t \phi(s_t). \tag{D.1}$$

We interpret the scores given by the human expert as noisy estimates of the discounted sum of rewards of the associated trajectories. The problem thus consists in regressing the scores v_i on the mappings of the histories $\mu(h_i)$. Formally, we require that the underlying procedure asymptotically minimizes the risk based on the classic ℓ_2-loss:

$$\mathcal{R}(\theta) = \mathbb{E}[(V - \theta^\top \mu(H))^2]. \tag{D.2}$$

The joint distribution on V (scores) and $\mu(H)$ (mappings of histories) is imposed, it is the distribution used to sample the dataset $\mathcal{D}$. Given any procedure that asymptotically minimizes risk (D.2), and denoting by θ_n the estimate computed from the dataset $\mathcal{D}$, we obtain an estimate r_{θ_n} of the reward. For example, the distance minimisation algorithm estimates the reward by solving the following linear least-squares problem:

$$\theta_n = \operatorname*{argmin}_{\theta \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} \Big(v_i - \theta^\top \mu(h_i) \Big)^2 \tag{D.3}$$

$$= \left(\frac{1}{n} \sum_{i=1}^{n} \mu(h_i) \mu(h_i)^\top \right)^{-1} \frac{1}{n} \sum_{i=1}^{n} \mu(h_i) v_i, \tag{D.4}$$

assuming that the matrix $\frac{1}{n} \sum_{i=1}^{n} \mu(h_i)\mu(h_i)^\top$ is invertible (ordinary least-squares). If this is not the case, one can use for example ℓ_2-regularization [Tikhonov, 1963] or ℓ_1-regularization [Tibshirani, 1996], among others. Hence, given a set of trajectories scored by a human expert, we have a reward estimate r_{θ_n}. In the next section, we analyze the quality of this reward.

D.2 Analysis

To study the estimate r_{θ_n}, we assume that the scores provided by the expert correspond to a discounted cumulative sum of rewards, up to some noise, for a reward function that lies in the hypothesis space $\mathcal{H} = \{\theta^\top \phi(s), \theta \in \mathbb{R}^d\}$.

Assumption 1 *There exists a vector parameter $\theta_* \in \mathbb{R}^d$ and a centered noise η such that for any trajectory $h = (s_t)_{t=0}^T$ and any associated score v, one has $v = \sum_{t=0}^T \gamma^t r_{\theta_*}(s_t) + \eta(h)$.*

Under this assumption, the minimizer of risk (D.2) is

$$\theta_* = \operatorname*{argmin}_{\theta \in \mathbb{R}^d} \mathcal{R}(\theta) = A^{-1} b, \tag{D.5}$$

$$\text{with } A = \mathbb{E}[\mu(H)\mu(H)^\top] \text{ and } b = \mathbb{E}[\mu(H)V], \tag{D.6}$$

assuming that the matrix A is invertible. First, we assume that the estimator θ_n satisfies (with high probability) that $\mathcal{R}(\theta_n) \leq \mathcal{R}(\theta_*) + \epsilon$, for some error ϵ. Then, we instantiate the related results in the case of ordinary least-squares.

D.2.1 Propagation of errors

First, we want to control the risk $\mathbb{E}_\nu[(r_{\theta_*}(S) - r_{\theta_n}(S))^2]$ for some distribution ν over states.

Theorem 1 (Propagation of errors: rewards) *Write λ_m the minimum eigenvalue of $\mathbb{E}[\mu(H)\mu(H)^\top]$ and λ_M the maximum eigenvalue of $\mathbb{E}_\nu[\phi(S)\phi(S)^\top]$. Assume that θ_n satisfies $\mathcal{R}(\theta_n) \leq \mathcal{R}(\theta_*) + \epsilon$, then the associated reward satisfies*

$$\mathbb{E}_\nu[(r_{\theta_*}(S) - r_{\theta_n}(S))^2] \leq \frac{\lambda_M}{\lambda_m}\epsilon. \tag{D.7}$$

Theorem 1 shows that if ϵ is small, the estimated reward r_{θ_n} will be close to the true reward r_{θ_*}. However, we are ultimately interested in computing an optimal policy $\pi_*^{r_{\theta_n}}$ (optimal policy relatively to the reward r_{θ_n}) for the control problem at hand. The question we answer now is: how close to optimal is the policy $\pi_*^{r_{\theta_n}}$ respectively to the one of the true reward $\pi_{r_{\theta_*}}^*$? To do so, we bound the quantity $\mathbb{E}_{\nu'}[(v_*^{r_{\theta_*}}(S) - v_{\pi_*^{r_{\theta_n}}}^{r_{\theta_*}}(S))^2]$, states being sampled according to an arbitrary distribution ν' of interest (possibly different from the distribution ν).

Before stating the result, we provide some notations. For a distribution ν and a function $f \in \mathbb{R}^S$, we write $\nu f = \mathbb{E}_\nu[f(S)]$. For a stochastic kernel Q, we have $[Qf](s) = \mathbb{E}_{Q(.|s)}[f(S')]$. Therefore, νQ is a distribution such that $\mathbb{E}_{\nu Q}[f] = \mathbb{E}_\nu[Qf] = \mathbb{E}_{S\sim\nu}[\mathbb{E}_{S'\sim Q(.|S)}][f(S')]$.

Theorem 2 (Propagation of errors: values) *For a policy π, define C_π as the smallest coefficient such that $(1-\gamma)\nu'(I-\gamma P_\pi)^{-1} \leq C_\pi \nu$. Assume that θ_n satisfies $\mathcal{R}(\theta_n) \leq \mathcal{R}(\theta_*) + \epsilon$, then the associated value satisfies*

$$\mathbb{E}_{\nu'}[(v_*^{r_{\theta_*}}(S) - v_{\pi_*^{r_{\theta_n}}}^{r_{\theta_*}}(S))^2] \leq \frac{2\left(C_{\pi_*} + C_{\pi_*^{r_{\theta_n}}}\right)}{(1-\gamma)^2}\frac{\lambda_M}{\lambda_m}\epsilon. \tag{D.8}$$

Theorem 2 shows that, if the error ϵ is small, the optimal policy respectively to the learned reward function will be close to the optimal policy respectively to the unknown reward (closeness being measured in term of value functions). There is an additional multiplicative term compared to Th. 1. The concentrability coefficient C_π measures the dissimilarity between the distribution of data ν and the distribution $(1-\gamma)\nu'(I-\gamma P_\pi)^{-1}$, the γ-weighted occupancy measure induced by policy π when the initial state is sampled from ν' (the distribution of interest for controlling the value function) and the coefficient $(1-\gamma)^{-1}$ is the average optimization horizon. Both terms are standard when bounding value functions.

D.2.2 A finite sample analysis

In this section, we provide a finite sample analysis in the case where $\theta_n \in \mathbb{R}^d$ is the ordinary linear least-squares estimate of Eq. (D.4), based on the bound of Hsu et al.. For this, we need some technical assumptions.

Assumption 2 *The noise is subgaussian: there exists $\sigma \geq 0$ such that, almost surely, for all $\lambda \in \mathbb{R}$, $\mathbb{E}[e^{\lambda\eta(H)}|H] \leq e^{\frac{\lambda^2\sigma^2}{2}}$. Moreover, there exists a finite $\rho \geq 1$ such that, almost surely, $\frac{\|A^{-\frac{1}{2}}\mu(H)\|_2}{\sqrt{d}} \leq \rho$.*

The subgaussian assumption is easily satisfied if the scores provided by the expert and the basis functions are bounded (in this case, the noise is bounded, and a bounded random variable is subgaussian [Hoeffding, 1963]). The other assumption corresponds to condition 1 of Hsu et al.. For example, if for any $s \in \mathcal{S}$ we have $\|\phi(s)\|_\infty \leq \phi_{\max}$, then almost surely we have $\|\mu(H)\|_2 \leq \frac{\sqrt{d}}{1-\gamma}\phi_{\max}$ and the assumption holds for $\rho \geq \frac{\phi_{\max}}{(1-\gamma)\sqrt{\lambda_m}}$.

Corollary 1 *Let δ be such that $\ln \frac{9}{\delta} > \max(0, 2.6 - \ln d)$. If assumptions 1 and-2 hold and if $n \geq 6\rho^2 d \ln \frac{9d}{\delta}$, then with probability at least $1-\delta$ we have*

$$\mathbb{E}_\nu[(r_{\theta_*}(S) - r_{\theta_n}(S))^2] \leq \tag{D.9}$$

$$\frac{\lambda_M}{\lambda_m} \frac{\sigma^2(d + 2\sqrt{d \ln \frac{9}{\delta}} + 2 \ln \frac{9}{\delta})}{n} + o\left(\frac{1}{n}\right). \tag{D.10}$$

Under the same assumptions, w.p. at least $1-\delta$:

$$\mathbb{E}_{\nu'}[(v_*^{r_{\theta_*}}(S) - v_{\pi_*^{r_{\theta_n}}}^{r_{\theta_*}}(S))^2] \leq \tag{D.11}$$

$$\frac{2\left(C_{\pi_*} + C_{\pi_*^{r_{\theta_n}}}\right)}{(1-\gamma)^2} \frac{\lambda_M}{\lambda_m} \frac{\sigma^2(d + 2\sqrt{d \ln \frac{9}{\delta}} + 2 \ln \frac{9}{\delta})}{n} + o\left(\frac{1}{n}\right). \tag{D.12}$$

This result mainly tells that in the case of ordinary least-squares, we have $\epsilon = O(\frac{d}{n})$, and thus the same rate for the errors of the reward and of the value (of the optimal policy), up to the additional constants arising because of error propagation. Since d is the number of features and n the number of trajectories, this gives an idea of the amount of data required to attain a given accuracy (however, notice that the concentrability coefficients can be hardly estimated, a standard problem in reinforcement learning). A wiser (but much more difficult) analysis would take into account the number of transitions instead of the number of trajectories, yet this is beyond the scope of this document.

D.3 Experiments on Garnets

D.3.1 Garnets

Testing distance minimisation requires three key elements. First, to build a finite MDP, which means an MDP with a finite set of states and actions. Second, to generate trajectories of random lengths in this MDP. And finally, to score these trajectories with a given noise η.

Garnets [Archibald et al., 1995] are an abstract class of finite MDPs, easy to build. Here, we consider a special case of Garnets specified by three parameters: $(N_\mathcal{S}, N_\mathcal{A}, N_B)$. Parameters $N_\mathcal{S}$ and $N_\mathcal{A}$ are respectively the number of states and of actions. Thus, $\mathcal{S} = (s_i)_{i=1}^{N_\mathcal{S}}$ and $\mathcal{A} = (a_i)_{i=1}^{N_\mathcal{A}}$ are, respectively, the state and action spaces. The parameter N_B ($N_B \leq N_\mathcal{S}$), called the branching factor, defines for each couple (s,a) the number of next states. N_B states are drawn uniformly and without replacement from $\mathcal{S}$ and form the set of next states of (s,a) noted $\mathcal{S}_{s,a} = (s'_i)_{i=1}^{N_B}$. Then, to define the transition probabilities P, for each state-action (s,a), we draw randomly and uniformly in $[0,1]$ $N_B - 1$ cutting points. Let us note $(p_i)_{i=1}^{N_B-1}$ this set of cutting points ranged in increasing order, $p_0 = 0$ and $p_{N_B} = 1$. To completely define the dynamics, one assigns $P(s'_i|s,a)$ according to the following rule: $\forall i \in \{1, \ldots, N_B\},\ P(s'_i|s,a) = p_i - p_{i-1}$. Finally, for each state $s \in \mathcal{S}$, the reward $r(s)$ is drawn randomly and uniformly in $[0,1]$ and $\gamma = 0.9$. This choice imposes that the non-perturbed score of a trajectory $h = (s_t)_{t=0}^T$ is bounded as follows $0 \leq \sum_{t=0}^T \gamma^t r(s_t) \leq \frac{1}{1-\gamma} = 10$. As we choose finite MDPs, a canonical choice of features ϕ is the tabular basis $\phi : \mathcal{S} \to \mathbb{R}^{N_\mathcal{S}}$ where $\phi(s) \in \mathbb{R}^{N_\mathcal{S}}$ is a vector which is null excepted in s where it is equal to 1.

The strategy chosen to generate a trajectory h of random length in a Garnet following a policy π consists in first choosing randomly and uniformly a starting state $s_0 \in \mathcal{S}$. Then, starting from s_0 we apply, with

probability $(1-p)$ ($p \in (0,1)$), the policy π to the current state s_i in order to go to the next state s_{i+1} and with probability p we stop the trajectory. Doing so, we obtain a trajectory $h = (s_t)_{t=0}^{T}$ where the length of the trajectory $T+1$ is a geometrically distributed random variable ($\forall k \in \mathbb{N}^*, Pr(T=k) = (1-p)^{k-1}p$, $\mathbb{E}[T] = \frac{1}{p}$ and $Var[T] = \frac{1-p}{p^2}$). We choose this strategy because it is easy to implement and allows us to control the mean of the length of the trajectories.

Finally, to give a score to a trajectory $h = (s_t)_{t=0}^{T}$, we proceed as follows. Let η be a Gaussian distributed and real valued random variable with mean $\nu = 0$ and variance σ^2 ($\eta \sim \mathcal{N}(0, \sigma^2)$), the score of the trajectory h is $v(h) = \lceil \sum_{t=0}^{T} \gamma^t r(s_t) + \eta \rfloor = \sum_{t=0}^{T} \gamma^t r(s_t) + \eta + \eta_q$, where $\lceil x \rfloor$ is the nearest integer from x and η_q is a quantification noise which is supposed to be a uniformly distributed random variable of mean $\mu = 0$ and variance $\sigma^2 = \frac{1}{12}$. We deliberately add a quantification noise in order to model the fact that experts are often asked to use an integer scale.

D.3.2 Results

So as to evaluate the distance minimisation performance, we first want to measure the mean performance of this algorithm over several Garnets when the number of trajectories, N_T, grows. To do so, we create $(G_q)_{q=1}^{N_G}$ Garnets of size $(N_{\mathcal{S}} = 100, N_{\mathcal{A}} = 5, N_B = 10)$ where we compute an optimal policy π_q^e w.r.t the real reward r_q of G_q via the policy iteration algorithm. For each G_q and each iteration $j \in \{1, \ldots, N_{It}\}$ ($N_{It} = 10$), we generate N_T trajectories $(h_i)_{i=1}^{N_T}$ according to a random policy π_r (at each state s, the probability to choose action a is $\frac{1}{N_{\mathcal{A}}}$) with $p = 0.01$ such that the mean length is around 100. We compute, for each trajectory the score $v(h_i)$ with $\eta \sim \mathcal{N}(0, \sigma^2 = 1)$. Here, the standard deviation of the noise represents 10% of the maximum non perturbed-score achievable by a trajectory. Thus, for each G_q, each iteration j and a given number of trajectories N_T, we obtain a set $(h_i, v_i = v(h_i))_{i=1}^{N_T}$ that we use as an input for distance minimisation which outputs the reward $r_{q,j}$. If $\pi_*^{r_{q,j}}$ is the optimal policy w.r.t $r_{q,j}$, then the performance of distance minimisation is measured via the normalized error between the value functions (w.r.t r_q) of π_q^e and $\pi_*^{r_{q,j}}$: $T_{q,j}(N_T) = \|v_{\pi_q^e}^{r_q} - v_{\pi_*^{r_{q,j}}}^{r_q}\|_2 \|v_{\pi_q^e}^{r_q}\|_2^{-1}$, where $\|.\|_2$ is the Euclidean norm. The lower the error is, the better is the performance. Finally, the mean error $T(N_T)$ is the mean of the $(T_{q,j}(N_T))_{1 \le q \le 50}^{1 \le j \le 10}$. In Fig. D.1, we plot the distance minimisation error, $T(N_T)$, where N_T varies from 100 to 1000 and the error corresponding to the random policy. We observe, as predicted by the analysis, that the error converges to zero as N_T grows.

Another interesting aspect to evaluate is the tolerance of distance minimisation to the noise $\eta \sim \mathcal{N}(0, \sigma)$ over a number of Garnets. To do so, we realize exactly the same experiment as before except that N_T is fixed to 500 and σ (standard deviation of the noise) is now the varying parameter. For each G_q, each iteration j and for a given σ, distance minimisation outputs the reward $r_{q,j}$ and the error $T_{q,j}(\sigma)$ of distance minimisation is defined as before. The mean error $T(\sigma)$ is the mean of the $(T_{q,j}(\sigma))_{1 \le q \le 50}^{1 \le j \le 10}$. In Fig. D.2, we plot the distance minimisation error, $T(\sigma)$, where σ varies from 0 to 5 which represents 50% of the maximum non-perturbed score given to a trajectory. In this plot, the distance minimisation algorithm is referred to as SBIRL. We observe that the performance is good when the noise is low and deteriorates as the noise gets higher. We also remark that the standard deviation of the error, which appears as a shade in Fig. D.2, gets higher as the noise increases.

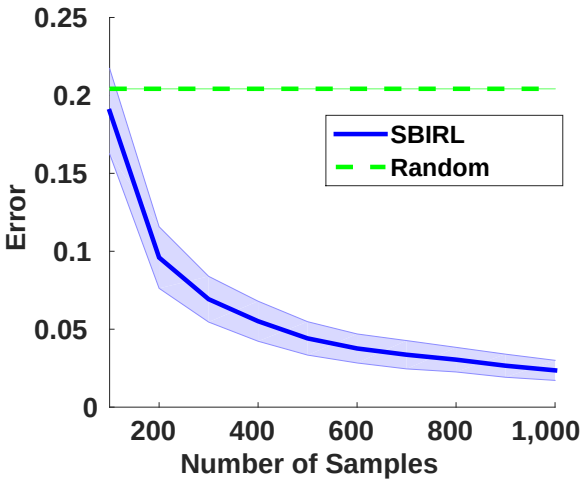


Figure D.1: Distance minimisation error as N_T grows

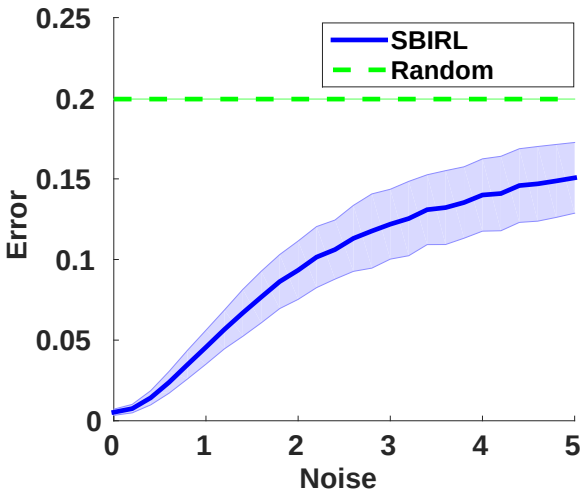


Figure D.2: Distance minimisation error as σ grows