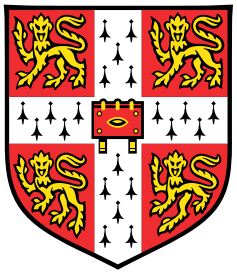
Hierarchical Dialogue Management



Francesca Giordaniello

Department of Engineering
University of Cambridge

This dissertation is submitted for the degree of
Master of Philosophy

Pembroke College 11 August 2017

Declaration

I, Francesca Giordaniello of Pembroke College, being a candidate for the M.Phil in Machine Learning, Speech and Language Technology, hereby declare that this report and the work described in it are my own work, unaided except as may be specified below, and that the report does not contain material that has already been used to any substantial extent for a comparable purpose.

Total word count: 12547

Francesca Giordaniello
11 August 2017

Acknowledgements

I would like to thank my supervisors Milica Gašić and Thomas Voice for the always inspiring meetings and their guidance and support in the development of the project.

I am also thankful to Iñigo Casanueva and Stefan Ultes for their willingness and the extremely useful aid in dealing with the *cued-pydial* framework.

Abstract

The dialogue management in a Spoken Dialogue System aims at learning a decision rule that can select the best system action in response to the user goal. However, this task results extremely hard for on-line applications, especially because these systems usually act in a multi-domain setting and require a large amount of training data. Additionally, the environment they have to perform in is generally highly noisy, because of the variability of human speech that causes errors in the transcriptions. As a consequence, the input received by the system is corrupted and the response provided to the user is consequently affected. Moreover, the uncertainty on the user intention generates a large input space, since every possible utterance needs to be considered. Therefore, the computational cost dramatically increases and contributes to the complexity of the optimisation problem. In this work, we try to exploit the similarities between the domains and improve the generalisation capability of the model when a large set of domains is considered. For this purpose, we build a tree-structured hierarchy for organising the domains, so that similar domains are clustered together. We show that the approach proposed provides performances that are generally comparable to the most recently used models in the literature. Furthermore, the slight improvements gained in some of the experiments open to other developments for future works.

Table of contents

Introduction **1**

Background **3**

2.1 Statistical Spoken Dialogue Systems 3

2.1.1 Spoken Language Understanding 3

2.1.2 Ontology 4

2.1.3 Dialogue Manager 5

2.1.4 Natural Language Generation 7

2.1.5 User simulator 7

2.2 Reinforcement Learning 7

2.2.1 Markov Decision Processes 7

2.2.2 Partially Observable Markov Decision Processes 11

2.2.3 Gaussian Processes for RL 13

2.3 Multi-domain Management 17

2.3.1 Bayesian Committee Machine 19

2.3.2 Hierarchical Mixture of Experts 20

Methods **23**

3.1 Hierarchical Bayesian Committee Machine 23

3.1.1 General approach 23

3.1.2 Automatic model selection 25

Experiments **29**

4.1 Experimental setup 29

4.2 In-domain and BCM comparison 30

4.3 BCM and HBCM comparison 31

4.3.1 Small ontology 31

4.3.2 Large ontology 32

4.4 Automatic model selection 34
4.5 Domains contribution 36

Conclusion **39**
5.1 Discussion 39
5.2 Future work 40
5.2.1 Greedy architecture selection 40
5.2.2 Parallelisation 41
5.2.3 Avoiding sparse approximation 41

References **43**

Appendix A **45**

Introduction

Spoken Dialogue Systems (SDSs) represent an intuitive tool for an easy user-machine interaction thanks to the exploiting of natural language as communication medium. This has brought a progressive interest in the field and in the search for further improvements and more efficient methods that can allow the spreading of this technology in various day-life applications.

An SDS is a modular architecture in which all the components carry out hard Machine Learning tasks and can be developed individually. In this work, we mainly focus on the Dialogue Manager unit, specifically on the policy optimisation problem. The design of SDSs represents a complex problem and the solution for the policy learning task still needs to be improved, despite the recent advances in multi-domain management.
The issues to be addressed are multiple and varied. First of all, the environment in which SDSs are used is generally noisy since human speech is highly variable, for example in terms of voice characteristics (e.g. accent, tone) and occurrence of unseen inputs (e.g. new words or domains). Therefore, the model needs to be robust to possible speech understanding errors produced during the user's spoken sentences recognition and should include a measure of this uncertainty. Moreover, SDSs should be able to converse about multiple topics, although this may represent an expensive feature. In fact, a large amount of data would be necessary for each of them. On the contrary, it may happen to have a wide discrepancy on the number of dialogues occurring on each domain, especially when freely available datasets are used. In these cases, guaranteeing the same amount of data for every domain considered might be very costly. For this reason, the system is desirable to be able to balance the lack of information on one domain with the more reliable data available on other domains.

Recent studies proposed several solutions for these problems. For instance, Partially Observable Markov Decision Processes (POMDPs) have been formulated to cope with the uncertainty on the system state and observations. In addition, Gaussian Processes (GPs) haven proved to be a powerful tool for data reuse, since their Bayesian non-parametric nature

allows to infer on unseen regions of the input space, as well. However, besides the important steps forward in policy optimisation, SDS tasks remain computationally expensive in a multi-domain set up, and they require sparse approximation methods to be solved.

We propose to structure the policy using a hierarchy over the domains that it covers, in order to exploit the generalised information that lies between related domains and boost the overall performance. Furthermore, this architecture opens the possibility for scaling to a wider set of domains and reducing the computational cost. The latter, in particular, can be achieved by parallelising the operations or overcoming the need for sparse approximations.
The practical implementation of the system is fully performed with the *cued-pydial* framework, which is entirely developed within the Dialogue Systems Group at the Cambridge University Engineering Department. Besides the SDS management, the tool kit also provides a large set of domains and the possibility to use a simulated user in order to test the model. As part of this project, some further modules have been implemented for developing the methods proposed.

Background

2.1 Statistical Spoken Dialogue Systems

A Statistical Spoken Dialogue System (SSDS) is an architecture composed by several independent modules, each covering a specific role in the interaction with the user. In Figure 2.1 we illustrate the different units, which are described in the following sections.

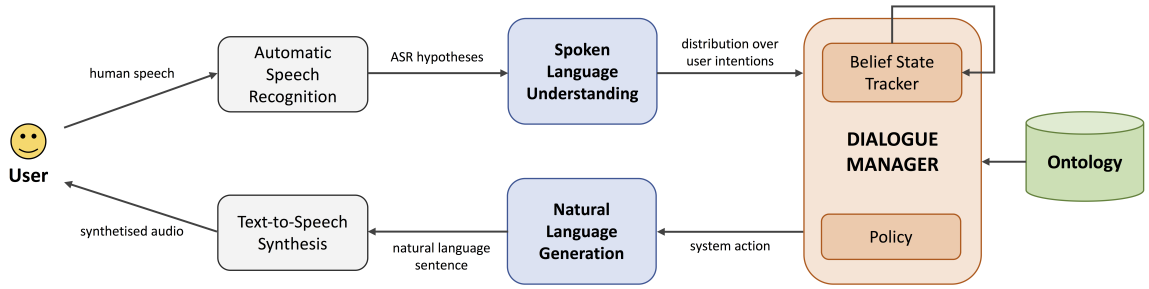


Figure 2.1. Modular architecture of a Statistical Spoken Dialogue System.

2.1.1 Spoken Language Understanding

The spoken query provided by the user needs to be transformed into a suitable representation that is understandable by the system. For this purpose, the sentence waveform is processed through Automatic Speech Recognition (ASR) and converted into a set of N-best transcriptions. Then, the Spoken Language Understanding unit (SLU) decodes these hypotheses into a semantic representation (Figure 2.2).

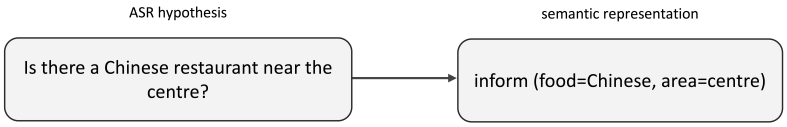


Figure 2.2. Example of ASR hypothesis conversion into its semantic decoding.

2.1.2 Ontology

The set of domains and their properties the SDS and the user can talk, ask and provide information about is known as the ontology. These are gathered in the form of a structured database in the *cued-pydial* framework. The characteristics of every domain define the structure of the database and represent the ontology schema. In all the experiments we assume that each dialogue concerns only one single domain, that is the goal of the user cannot change within a dialogue.

The database fields represent the properties and information available for a certain domain (*slots*), such as address, pricerange, food type and so on. Each entity of a certain domain is stored as a row in the database and the fields are filled with the correspondent slots value. We say that a slot is informed by the user when a value for it is assigned. The slots are divided in two semantic classes, namely *requestable* and *informable slots*. The requestable slots represent those properties that can be used by the user to constrain the search, such as the type of *food* for the selection of a restaurant. On the other hand, slots are called informable when they are only queried to provide further information on a specific entity to the user. For example, the *phone* or *address* of a restaurant cannot be informed by the user nor used to look for a restaurant directly. In particular, requestable slots are of main importance because they are used by the system to discern the entities that might match the user goal.

The domains available in *cued-pydial* and used for this study are the following:

SFR:	restaurants in San Francisco
CR:	restaurants in Cambridge
SFH:	hotels in San Francisco
CH:	hotels in Cambridge
L6:	laptops, 6 requestable properties
L11:	laptops, 11 requestable properties
CS:	shops in Cambridge
CA:	attractions in Cambridge

CT:	transport in Cambridge
TSBp:	video players
TV:	televisions

Refer to Figure A.1 in Appendix A for a list of all the slots in each domain.

2.1.3 Dialogue Manager

The Dialogue Manager (DM) is the core of the system behaviour management. It is composed by two units, the *belief state tracker* and the *policy*. The former updates the state of the system basing on the observation received from the SLU, whereas the latter represents the decision rule according to which it is able to select the best action to take at each turn.

Dialogue Belief State Tracker

The dialogue belief state tracker maintains the information on the user's requests and keeps track of the system outputs and context from the previous turns. The uncertainty of the system about the intent of the user is expressed as a probability distribution over the possible states called the *belief state*. In particular, each state is decomposed into three sources of information, namely the *goal constraint*, the *requested slots* and the *search method*. This results in a smaller number of conditional dependencies, and therefore it reduces the number of states the probability distribution is computed over [1]. The goal constraint assigns to each slot a value chosen by the user and represents the information the user has required for the task up until the current turn. The requested slots correspond to the confidence on the user requesting a particular slot. Finally, the method indicates the modality in which the slots are informed by the user, that is how it interacts with the system.

In this work, the state is modeled as a *focus tracker*, which accumulates the history and updates the model belief state basing on the information gathered from the previous turns of the dialogue up to this turn and from the current observation. Specifically, the update at time t is formulated as:

$$b_t(s_t = s) = \left(1 - \sum_{s'} o(s')\right) b_{t-1}(s_{t-1} = s) + o(s) \tag{2.1}$$

where $b_t(s_t = s)$ represents the probability that the state at time t is s (belief state), and $o(s)$ is the probability of the observation relating to state s. In other words, the focus tracker is able to balance the contribution of the past history b_{t-1} depending on how much informative the observations in each state are.

Dialogue Policy

The current belief state is fed into the policy optimiser in order to select the best action to take among all the possible ones stored in the ontology (*master action set*). Therefore, the complete feature space would contain the combination of belief space and master action space, that is all the possible actions on all the available values. However, in order to reduce the number of actions to be enumerated, the master action space is projected into a smaller set, called *summary action space* $\mathcal{A}$. The latter only contains the actions available for the requestable and informable slots of a particular domain, namely [2]:

- *request_*$\langle$*requestable-slot*$\rangle$: request properties required by the user.
- *confirm_*$\langle$*requestable-slot*$\rangle$: confirm the intention of the user.
- *select_*$\langle$*requestable-slot*$\rangle$: ask clarification when uncertain between two possible user intentions.
- *inform*: inform the value of a slot, according to different methodologies:
  - *inform_byname*: the slot value of a particular entity explicitly named by the user is informed.
  - *inform_alternatives*: provide alternative venues for which the slot value probabilities are higher than a certain threshold; otherwise, inform the user that there are no entities matching the properties desired.
  - *inform_requested*: the requested slots are informed with the correspondent values from the last informed entity.

The summary action and the current belief state are then passed to the *summary function*, which combines these two inputs and maps them into the corresponding master action.

The policy optimisation is implemented through the *GPSARSA* algorithm, which applies the advantages of Gaussian Processes-based methods to the SARSA algorithm. The method is explained in detail in Section 2.2.3

2.1.4 Natural Language Generation

The system act selected is then fed into the Natural Language Generation (NLG). In particular, it is encoded again into a sentence in natural language and Speech Synthesis techniques are used to generate the correspondent waveform. In this way, the user can receive the answer to their query in a spoken fashion, according to the general Text-To-Speech (TTS) procedure.

2.1.5 User simulator

In general, collecting a large corpora to train the model is an expensive and impractical procedure. Furthermore, the domains included in the dataset may not correspond to those characteristic of the SDS application. The solution to this problem is to build a *user simulator*, that resembles the human user behaviour and that can be use in the training and development processes.

The method that models the simulated user is the goal- and agenda-based state representation [3], that involves a sequence of Markovian state transitions and dialogue acts for the user, as well (see Section 2.2.1). The state, in particular, is factorised into a *goal* and an *agenda*, which are updated during the dialogue. The goal is randomly generated for every new dialogue and consists of the required slots to constrain the search and the desired information (e.g. *name*, *address*). The agenda is a stack-like architecture storing the user acts, which are removed when no longer relevant or added in response to system acts. In this way, it is also possible to keep track of the history of the current dialogue.

As illustrated in Figure 2.3, when a simulated user is used the DM is fed a distribution of abstracted utterances directly, without the intermediate action of the SLU. Specifically, an *error model* is introduced to simulate the uncertainty over user's intentions produced by the ASR unit. Similarly, the NLG component is not used since there is no need to synthetise speech in a human-understandable form.

2.2 Reinforcement Learning

2.2.1 Markov Decision Processes

The aim in a Reinforcement Learning (RL) task is to learn from the interaction with the environment in order to achieve a goal. This interaction is described in terms of states,

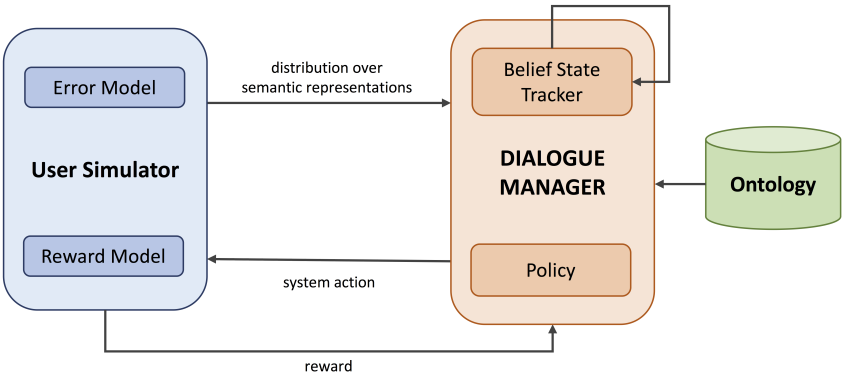


Figure 2.3. SDS components when a simulated user is used. The user simulator is also responsible for providing a reward to the system.

actions and rewards (Figure 2.4). In particular, the actions are chosen according to a decision rule called the *policy*. The policy is said to be deterministic when the action to be taken in every state is exactly predictable. Conversely, for stochastic policies not a fixed action but a probability distribution over actions is provided given a state: $\pi_t(a \mid s) = p(a_t = a \mid s_t = s)$. The latter are also of major interest in this study. The learning process involves finding the best trade-off between the *exploitation* of the actions that are known to produce a high reward and the *exploration* of the state-action space to discover better alternatives for the future selections [4].

We say that the model satisfies the *Markov property* when the state and reward at time step t only depend on the state at time t-1 and the action at time t-1 [4, 5], rather than on the whole sequence of states, actions and rewards. In other words, the following probability distributions are equal $\forall s', r, s_{t-1}, a_{t-1}$:

$$P(s_t = s', r_t = r | s_{t-1}, a_{t-1}, r_{t-1}, \ldots, s_0, a_0, r_0) = P(s_t = s', r_t = r | s_{t-1}, a_{t-1}) \tag{2.2}$$

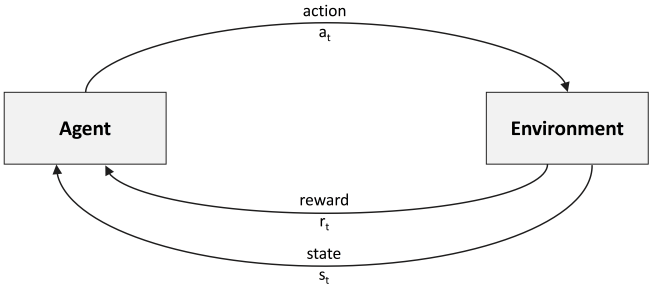


Figure 2.4. Agent-environment interaction. At each turn t the agent acts on the environment, obtaining a reward and changing its internal state given the observation of the environment .

Therefore, we can formulate the system as a Markov Decision Process (MDP), which is defined as a tuple $(\mathcal{S}, \mathcal{A}, \mathrm{T}, r)$ where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of possible actions, T is the transition probability $P(s_{t+1} \mid s_t, a_t)$ of reaching the state s_{t+1} when taking the action a_t from the state s_t, and r_t is the immediate reward for that action from that state.

Given this definition of the model, we can now express the formulation of the policy optimisation, which provides a solution for the MDP problem. First of all, we define the *return* R_t from time t, which represents the "desiderability" of the state at time t:

$$R_t = \sum_{k=0}^{T-t-1} \gamma^k r_{t+k+1} \tag{2.3}$$

where γ is the discount factor, with $0 \leq \gamma \leq 1$, and r_i is the immediate reward gained by the agent in the i-th state. The discount factor is used in *continuing tasks* when the user-system interaction is continuous and repeated for an infinite number of steps ($T \to \infty$). In this way, we ensure that the total reward has a finite value. However, SDSs generally involve a fixed-duration interaction defined by the number of utterances in the dialogue (*episodic task*). As a consequence, we will set $\gamma = 1$ in the implementation and simply assign a positive value to r_t when the t-th dialogue is successful and a negative value otherwise.

The behaviour of the system is led by the learning of the *policy* π, that represents the decision rule according to which the best action is performed. Therefore, we define the *state-value function* $V_\pi(s)$ of a state s for a certain policy π as an indication of how good that state is for the agent in the long term, by taking into account the sequence of states it is likely to follow when the policy π is applied. This relation is expressed as the expected cumulative reward starting from the state s and following the policy π:

$$V_\pi(s) = \mathbb{E}_\pi\{R_t \mid s_t = s\} \tag{2.4}$$

Similarly, the *action-value function* $Q_\pi(s,a)$ defines the value of taking the action a under the policy π when the system state is s. In practice, it represents the expected return given the agent is in s, performs a, and follows the policy π thereafter:

$$Q_\pi(s,a) = \mathbb{E}_\pi\{R_t \mid s_t = s, a_t = a\} \tag{2.5}$$

Finally, the policy is defined as to maximise the expected cumulative reward. In particular, since value functions define a partial ordering over policies, there is always at least one policy

that is better than or equal to all the others:

$$\pi^*(s) \geq \pi(s) \iff V_{\pi^*}(s) \geq V_\pi(s) \quad \forall s \in \mathcal{B} \tag{2.6}$$

Therefore, the optimal policy π^* is:

$$\begin{aligned} \pi^*(s) &= \underset{\pi}{\operatorname{argmax}} V_\pi(s) \\ &= \underset{\pi}{\operatorname{argmax}} Q_\pi(s,a) \end{aligned} \tag{2.7}$$

Different methods are possible regarding the way policies are improved. Specifically, in *off-policy* approaches separated policies are maintained, respectively to guide the system actions and to be evaluated. In other words, the policy used to select the action in a certain region of the space differs from the one that has been trained, because the latter is integrated with the new data collected. Conversely, in *on-policy* methods the policy that is improved is used for decision making, as well. In this way, the agent is mainly induced to find the policy that guarantees the exploration of the space [4].

A further differentiation concerns the assumptions on the knowledge about the MDP problem, in particular leading to a *model-based* or a *model-free* approach. In the former we assume that an explicit model of the environment is known, and therefore the agent is able to always identify the state it is in, the consequences of its actions and the correspondent rewards [6]. As a result, the optimal value functions can be evaluated given the knowledge of $\mathcal{S}$, $\mathcal{A}$, T and r through the *Bellman equations*:

$$V_{\pi^*}(s) = \max_a Q_{\pi^*}(s,a) = \max_a \sum_{s'} T(s'|s,a)[r_{t+1} + \gamma V_{\pi^*}(s')] \tag{2.8}$$

$$Q_{\pi^*}(s,a) = \sum_{s'} T(s'|s,a)[r_{t+1} + \gamma \max_{a'} Q_{\pi^*}(s',a')] \tag{2.9}$$

This kind of methods usually infer the optimal policy through iterative updates of V, by exploiting Dynamic Programming (DP) algorithms (e.g. *value iteration* and *policy iteration* algorithms, see [4]). These algorithms perform a full backup in each iteration, that is the value function is updated for every state basing on the old value of its successor states. However, they result in computationally expensive procedures since they operate on the entire state space, whose size grows exponentially for the nature of the problem.

On the contrary, a model-free approach provides a limited knowledge for the agent, which has no information on the environment and does not need the transition model to perform the updates. In this case, the agent rather learns from experiencing and interacting with the environment. For instance, Temporal-Difference (TD) methods learn fully from their experience by immediately updating the value function V after each step, basing on the current estimate $V(s_{t+1})$ and the observed reward r_{t+1}:

$$V(s_t) = V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)] \tag{2.10}$$

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{2.11}$$

where $\alpha > 0$ is the step size.

One example is the *SARSA* algorithm, an on-policy method which derives its name from the fact that it uses all the elements of the tuple of events $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$ for the update. Moreover, it is guaranteed to converge to an optimal policy if all the state-action pairs are visited an infinite number of times. This can be achieved with ε-*greedy* policies where epsilon converges to 0 over time. For instance, at each turn we could select the best action with probability ε and a random action otherwise.

Algorithm 1 Implementation of the SARSA algorithm.

Initialise Q arbitrarily
repeat
    Initialise s
    Choose a from s ε-greedily
    repeat
        Take action a, observe r and s'
        Choose a' from s' ε-greedily
        $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)]$
        $s \leftarrow s'$
        $a \leftarrow a'$
    until s is terminal
until convergence

2.2.2 Partially Observable Markov Decision Processes

The formulation of the problem described above as an MDP, however, does not optimally model real-life systems because the state of the agent is generally uncertain and the output

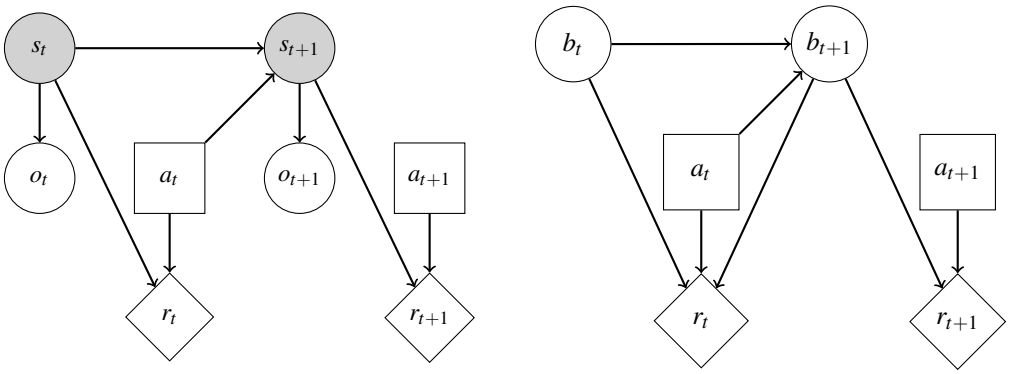
is affected by noise. Regarding SDS, in particular, a recent common approach involves the description of the DM as a Partially Observable Markov Decision Process (POMDP), in which the state at each time step is only partially observable and depends on observations that may be noisy [7]. As a consequence, whereas the actual sequence of states is still modeled by transition probabilities, a distribution over all possible states (i.e. user intentions) is maintained as *belief state*. In other words, we indicate with $b_t(s_t)$ the probability of being in the state s at time t. Moreover, since the process is Markov, the current belief state contains all the information about the past history and the initial state.

Therefore, a POMDP is formulated as a tuple $(\mathcal{S}, \mathcal{A}, \mathrm{T}, r, \mathcal{O}, \mathrm{Z}, \gamma, b_0)$ [1], where:

- $\mathcal{S}$ is the set of discrete states s_t that represent the actual intention of the user.
- $\mathcal{A}$ is the set of possible master actions a_t.
- T is the transition probability $P(s_{t+1}|s_t, a_t)$ of getting to the state s_{t+1} from the state s_t when performing the action a_t.
- $r(s_t, a_t)$ is the immediate reward for performing the action a_t when the system is in the state s_t.
- $\mathcal{O}$ is the set of observations o_t.
- Z is the observation probability $P(o_t|s_t, a_{t-1})$ of obtaining the output o_t given that the system state is s_t and the action a_{t-1} is performed.
- γ is the discount factor.
- b_t is the belief state at time t.

We illustrate in Figure 2.5a the formulation of the SDS as a POMDP model.

The ability to deal with uncertainty and retain the previous actions yields the number of states, actions and observations to grow dramatically. This results in the intractability of exact policy optimisation in POMDPs [1]. Different approaches are possible to overcome this issue, such as *tabular methods* or *function approximation*. The former assign each state-action pair to the entry of a vector or table and are only applicable for problems with a limited input space. On the other hand, the latter can handle much larger state sets. In fact, they are guaranteed to generalise by taking examples from a function (e.g. value function) and approximating its values for states that have not been visited yet [4]. A further approach we exploit in this work interprets the POMDP model as an MDP with a continuous state space [5]. Specifically, the sequence of states is now composed by the belief states that are observable, whereas we maintain the dependence of the reward on actions and belief states (Figure 2.5b).



(a) POMDP model. **(b)** SDS interpreted as an MDP model.

Figure 2.5. Graphical representation of the model described. White blocks represent observable variables, whereas the hidden variables are coloured in gray.

In this way, it is possible to combine the advantages of the belief state evolving as a Markov process and the reward-based policy optimised through reinforcement learning techniques. In other words, we introduce a Bayesian approach to model the uncertainty of the system and the optimisation of a policy aiming at maximising a reward. This guarantees a satisfactory level of robustness for the possible errors introduced by real-life application noise [1].

Given this definition of the model and the continuity of the state space in a POMDP, the value functions are now modified as follows:

$$V_\pi(b) = \mathbb{E}_\pi\{R_t \mid b_t = b\} = \int_{b'} p(b' \mid b, \pi(b))[r(b,a) + V_\pi(b')]db' \tag{2.12}$$

$$Q_\pi(b,a) = \mathbb{E}_\pi\{R_t \mid b_t = b, a_t = a\} = \int_{b'} p(b' \mid b, a)[r(b,a) + Q_\pi(b', \pi(b'))]db' \tag{2.13}$$

and the optimal policy π^* is:

$$\begin{aligned} \pi^*(b) &= \underset{\pi}{\operatorname{argmax}}\, V_\pi(b) \\ &= \underset{\pi}{\operatorname{argmax}}\, Q_\pi(b,a) \end{aligned} \tag{2.14}$$

2.2.3 Gaussian Processes for RL

A Gaussian Process (GP) is defined as a collection of random variables, any finite number of which have joint Gaussian distributions [8]. A GP is specified by a mean function $m(x)$ and a

covariance function or kernel $k(x,x')$:

$$f \sim \mathcal{GP}(m(x),k(x,x')) \tag{2.15}$$

Each point on the function is Gaussian distributed. Thus, given an input vector x we can fix the mean and covariance parameters and obtain a normal distribution:

$$\mu_i = m(x_i), \qquad \Sigma_{ij} = k(x_i,x_j) \tag{2.16}$$

$$f \sim \mathcal{N}(\boldsymbol{\mu},\boldsymbol{\Sigma}) \tag{2.17}$$

Every new observation "pins" a particular function from the distribution, consequently imposing a correlation on the unobserved sequence (Figure 2.6). In particular, the covariance function has a fundamental role in the GP prediction because it identifies the shape of the prior we assign to the input set. Additionally, it encodes our assumptions on the function we are learning and our confidence about the data predicted [9]. Furthermore, it expresses similarities and correlations between the data, properties that are especially important during testing. In fact, the closer two inputs x the more similar their correspondent target values, and therefore the more informative they are about each other.

In addition, GPs are a Bayesian non-parametric model, that is they are not restricted by the parameters chosen, although the mean and covariance functions can assume different formulations. Moreover, the posterior distribution presents a certain flexibility in its final shape, due to the ability of the GP to adapt as more data are collected. This behaviour is maintained even if the prior knowledge about the data structure is incorporated in the function and even after a large number of data points are observed. Thanks to these desirable properties, GPs can be exploited in RL for function approximation.

From the application of GPs to RL (GPRL) the *GPSARSA* algorithm is derived [10], which models the Q-function as a Gaussian process having a priori zero mean and kernel function defined over belief-action values:

$$Q(\mathbf{b},a) \sim \mathcal{GP}(0,k((\mathbf{b},a),(\mathbf{b},a))) \tag{2.18}$$

Taking into account the stochastic state transitions in the continuous-space MDP we previously defined, we can assess that the return R defined in Section 2.2.1 is now a random variable [7]. Moreover, given the definition of the Q-function in equation (2.5), the return

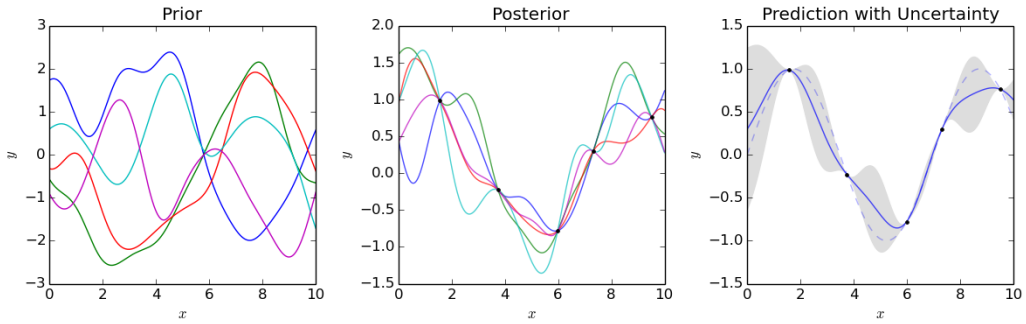


Figure 2.6. Gaussian Process Regression (prediction) with a squared exponential kernel. Left are draws from the prior distribution of functions. Middle are draws from the posterior predictive distribution. Right is the mean prediction with 1 standard deviation shaded. [1]

can be decomposed into a mean $Q_\pi(\mathbf{b},a)$ and a Gaussian noise residual $\Delta Q_\pi(\mathbf{b},a)$:

$$R_\pi(\mathbf{b}_t=\mathbf{b},a_t=a)=Q_\pi(\mathbf{b},a)+\Delta Q_\pi(\mathbf{b},a),\qquad \text{where } \Delta Q_\pi(\mathbf{b},a)\sim\mathcal{N}(0,\sigma^2) \tag{2.19}$$

Combining these assumptions with the recursive expression of the return:

$$R_t^\pi = r_{t+1}+\gamma R_{t+1}^\pi \tag{2.20}$$

we can write the relation between the immediate reward and the Q-function as:

$$r_{t+1}(\mathbf{b}_t=\mathbf{b},a_t=a)=Q_\pi(\mathbf{b},a)-\gamma Q_\pi(\mathbf{b}',a')+\Delta Q_\pi(\mathbf{b},a)-\gamma\Delta Q_\pi(\mathbf{b}',a') \tag{2.21}$$

where $\mathbf{b}_{t+1}=\mathbf{b}'$ and $a_{t+1}=a'$. The equation can be rewritten in its matrix form:

$$\mathbf{r}_t=\mathbf{H}_t\mathbf{q}_t^\pi+\mathbf{H}_t\Delta\mathbf{q}_t^\pi \tag{2.22}$$

where

$$\begin{aligned}\mathbf{r}_t &= [r_1,\ldots,r_t]^T\\ \mathbf{q}_t^\pi &= [Q_\pi(\mathbf{b}_0,a_0),\ldots,Q_\pi(\mathbf{b}_t,a_t)]^T\\ \Delta\mathbf{q}_t^\pi &= [\Delta Q_\pi(\mathbf{b}_0,a_0),\ldots,\Delta Q_\pi(\mathbf{b}_t,a_t)]^T\end{aligned}$$

[1] Figure and caption taken from https://commons.wikimedia.org/wiki/File%3AGaussian_Process_Regression.png.

$$\mathbf{H}_t = \begin{bmatrix} 1 & -\gamma & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \ddots & \ddots & \vdots & \vdots \\ 0 & \cdots & 0 & 1 & -\gamma \end{bmatrix}$$
$$\mathbf{B}_t = [(\mathbf{b}_0, a_0), \ldots, (\mathbf{b}_t, a_t)]^T$$

The posterior distribution of the Q-function is derived by applying GP regression given the set of belief-action $\mathbf{B}_t$ and reward $\mathbf{r}_t$ observations:

$$\begin{aligned} Q(\mathbf{b},a) \mid \mathbf{r}_t, \mathbf{B}_t &\sim \mathcal{N}(\bar{Q}(\mathbf{b},a), cov((\mathbf{b},a),(\mathbf{b},a))) \\ \bar{Q}(\mathbf{b},a) &= \mathbf{k}_t(\mathbf{b},a)^T \mathbf{H}_t^T (\mathbf{H}_t \mathbf{K}_t \mathbf{H}_t^T + \sigma^2 \mathbf{H}_t \mathbf{H}_t^T)^{-1} \mathbf{r}_t \\ cov((\mathbf{b},a),(\mathbf{b},a)) &= k((\mathbf{b},a),(\mathbf{b},a)) - \mathbf{k}_t(\mathbf{b},a)^T \mathbf{H}_t^T (\mathbf{H}_t \mathbf{K}_t \mathbf{H}_t^T + \sigma^2 \mathbf{H}_t \mathbf{H}_t^T)^{-1} \mathbf{H}_t \mathbf{k}_t(\mathbf{b},a) \end{aligned} \tag{2.23}$$

where

$$\begin{aligned} \mathbf{k}_t(\mathbf{b},a) &= [k((\mathbf{b}_0,a_0),(\mathbf{b},a)), \ldots, k((\mathbf{b}_t,a_t),(\mathbf{b},a))]^T \\ \mathbf{K}_t &= [\mathbf{k}_t(\mathbf{b}_0,a_0), \ldots, \mathbf{k}_t(\mathbf{b}_t,a_t)] \end{aligned}$$

The matrix $\mathbf{K}_t$ is the *Gram matrix* and can significantly grow as more observations are gathered, thus causing the intractability of the posterior computation. This issue is overtaken by applying the *kernel span sparsification method* [11]. Specifically, it approximates the Gram matrix by only considering a subset of *representative points* (collected in a *dictionary*) rather than the full belief-action space. The size m of the dictionary $\mathcal{D} = \{(\tilde{\mathbf{b}}_0, \tilde{a}_0), \ldots, (\tilde{\mathbf{b}}_m, \tilde{a}_m)\}$ is determined through the *sparsification threshold* ν, since we only maintain the points whose squared distance exceeds that value [10]:

$$k((\mathbf{b}_t,a_t),(\mathbf{b}_t,a_t)) - \mathbf{a}_t^T \tilde{\mathbf{k}}_{t-1}(\mathbf{b}_t,a_t) > \nu \tag{2.24}$$

where

$$\begin{aligned} \tilde{\mathbf{k}}_{t-1}(\mathbf{b}_t,a_t) &= [k((\mathbf{b}_t,a_t),(\tilde{\mathbf{b}}_0,\tilde{a}_0)), \ldots, k((\mathbf{b}_t,a_t),(\tilde{\mathbf{b}}_m,\tilde{a}_m))]^T \\ \mathbf{a}_t &= \tilde{\mathbf{K}}_{t-1}^{-1} \tilde{\mathbf{k}}_{t-1}(\mathbf{b}_t,a_t) \end{aligned}$$

in which $\tilde{\mathbf{K}}_t$ is the Gram matrix of the current set of representative points.

For every new pair of belief state and action a Gaussian distribution is defined, whose mean and covariance are given by equation (2.23). We can sample from these distributions to get

values for Q, so that the optimal policy from a certain belief state $\mathbf{b}$ corresponds to the action providing the highest Q-value:

$$\pi^*(\mathbf{b}) = \underset{a}{\operatorname{argmax}} \{\bar{Q}(\mathbf{b}, a) : a \in \mathcal{A}\} \tag{2.25}$$

In particular, in our MDP model the kernel function has to define correlations between points from both continuous and discrete spaces. Therefore, we decompose it into separate kernels over belief states and actions respectively:

$$k((\mathbf{b}, a), (\mathbf{b}, a)) = k_{\mathcal{B}}(\mathbf{b}, \mathbf{b}) \cdot k_{\mathcal{A}}(a, a) \tag{2.26}$$

Algorithm 2 Implementation of the GPSARSA algorithm.

```
Define a prior for Q
for all dialogues do
    Initialise b
    Choose a according to current Q estimate
    if (b, a) is representative then
        add (b, a) to dictionary
    end if
    for all turns do
        Take action a, observe r and b'
        Choose a' according to current Q estimate
        if (b', a') is representative then
            add to dictionary
        end if
        Update Q posterior mean and variance
        b ← b'
        a ← a'
    end for
end for
```

2.3 Multi-domain Management

In SDS applications the model usually has to be able to handle dialogues from a large number of domains, although the data available for the individual topics might not be sufficient for training an appropriate policy. For this reason, multi-domain management is a fundamental issue to be coped with, and many approaches have been developed for this purpose.

One approach is proposed in [12] and suggests the use of a generic model initially trained on all the available data. Then, as further data is collected for a certain domain, a specific policy can be trained, using the general model as prior knowledge.

A parallel approach involves the use of a Bayesian Committee Machine (BCM), that is exploited in this work, as well (see Section 2.3.1). Being first defined in [13], and then applied to SDSs in [14], it is based on the idea that every available domain is a committee member and contributes to the general policy depending on the uncertainty it carries.

From the description given in Section 2.2.3, it is clear that GPs can offer a valid tool to make efficient reuse of data and aid in overcoming the limited-data issue in multi-domain tasks. In fact, they guarantee the known portion of the space to influence and favor the exploration of the regions where unseen data points lie.

In GPRL the kernel function needs to be calculated between data points that may belong to different domains. Consequently, a method is needed to handle this computation if the samples differ in the number of slots (cardinality). In particular, as proposed in [14], the *normalised entropy* η is calculated for each slot s in every domain:

$$\eta(s) = -\sum_{v \in \mathcal{V}_s} \frac{p(s=v)\log p(s=v)}{|\mathcal{V}_s|} \tag{2.27}$$

where v is a value from the set $\mathcal{V}_s$ of possible values for s, and $p(s=v)$ is the probability that s takes that value. In this way, we have a measure of how useful each slot is. For example, consider the domain of hotels in Cambridge (CH) and assume all its instances are situated in the centre of the city ($area =$ "centre" for every entity in the CH database). Then, when we compute the normalised entropy for the slot *area*, it results that:

$$p(area = v) = \begin{cases} 1 & \text{if } v = \text{"centre"} \\ 0 & \text{if } v \neq \text{"centre"} \end{cases} \tag{2.28}$$

Therefore, since the logarithm is zero in the first case, we get a null contribution $\forall v \in \mathcal{V}_{area}$, that is $\eta(area) = 0$. This means that the slot *area* is not useful to the system to select the entity matching the user goal.
Afterwards, for requestable and informable semantic classes separately, the slots are sorted according to the decreasing value of their normalised entropy, and they are given an abstract name. The resulting slots for class c are then collected in a vector with elements $slot_1^c, slot_2^c, \ldots$ such that $\eta(slot_i^c) \geq \eta(slot_j^c)$ for $i \leq j$.

Finally, when the kernel function is computed for a pair of belief-action observations that differ in the number of slots (cardinality) the shortest vector is padded with zeros.

The procedure we described allows the policy to operate on data points that come from different domains and that may present a different number or non-matching slots. What is more, this method is completely free from human intervention and the kernel function that results is positive definite. Thus, the GP-based policy is well-defined in every data point [15].

2.3.1 Bayesian Committee Machine

The Bayesian Committee Machine (BCM) proposed in [13] allows to combine estimators that have been trained on different datasets. This approach is particularly suitable to facilitate the handling of large datasets and on-line learning, e.g. in GP regression. In this case, for instance, the computational cost for the inversion of the covariance function is cubic in the size of training data and causes the learning to be infeasible for big sets. The BCM method exploits several estimators, each of them focusing on a certain region of the space, and combines their predictions weighting them according to their confidence. Furthermore, by partitioning the dataset, also the computational cost is improved, since the complexity only grows linearly with the size of the training dataset [16].

Let us consider a predictive function f with additive Gaussian noise ε, so that our final predictions are: $g(x) = f(x) + \varepsilon(x)$. Given a training set of K input and target measurements (superscript m) $D = \{X^m, g^m\}$, with $X^m = \{x_1^m, \ldots, x_K^m\}$ and $g^m = (g_1, \ldots, g_K)^T$, we split it into M datasets $D = \{D_1, \ldots, D_M\}$. Similarly, the inputs and targets are partitioned as $X^m = \{X_1^m, \ldots, X_M^m\}$ and $g^m = \{g_1^m, \ldots, g_M^m\}$. Let us also consider a set of N query points (superscript q) $X^q = \{x_1^q, \ldots, x_N^q\}$ and the correspondent unknown responses $f^q = (f_1^q, \ldots, f_N^q)^T$. Let $\mathcal{D}_i = \{D_1, \ldots, D_i\}$ be composed by the subsets up to the i-th partition, with $i = 1, \ldots, M$. Then, we train a separate estimator on each of these subset.

Therefore, we can express the posterior predictive distribution as:

$$P(f^q \mid \mathcal{D}_{i-1}, D_i) \propto P(f^q) P(\mathcal{D}_{i-1} \mid f^q) P(D_i \mid \mathcal{D}_{i-1}, f^q) \tag{2.29}$$

We could approximate:

$$P(D_i \mid \mathcal{D}_{i-1}, f^q) \approx P(D_i \mid f^q) \tag{2.30}$$

if f^q contains the targets at the input locations of all the training data ($f^q \equiv f$), so that the outputs in the training data are independent. The approximation is also valid when f^q

determines f everywhere (N is large), when $\mathcal{D}_{i-1}$ and D_i are not highly correlated (e.g. inputs spatially separated) and the size of D_i is large (data more independent on average) [16]. We can apply the Bayes' theorem:

$$P(f^q \mid \mathcal{D}_{i-1}, D_i) \equiv const \times \frac{P(f^q \mid \mathcal{D}_{i-1}) P(f^q \mid D_i)}{P(f^q)} \tag{2.31}$$

and obtain the approximation of the predictive distribution:

$$\hat{P}(f^q \mid \mathcal{D}_{i-1}, D_i) = const \times \frac{\prod_{i=1}^{M} P(f^q \mid D_i)}{P(f^q)^{M-1}} \tag{2.32}$$

where *const* is a normalising constant and we divide M-1 times by the prior because we are multiplying posterior probabilities.

When Gaussian distributions are used, we can assume the prior density $P(f^q)$ has zero mean and covariance Σ^{qq}, and the posterior distribution from the i-th partition (committee member) has mean $\mathbb{E}(f^q \mid D^i)$ and covariance $cov(f^q \mid D^i)$. Then, the mean and covariance of the approximated distribution, resulting from the combined estimators, are respectively:

$$\hat{\mathbb{E}}(f^q \mid D) = \frac{1}{C} \sum_{i=1}^{M} cov(f^q \mid D_i)^{-1} \mathbb{E}(f^q \mid D_i) \tag{2.33}$$

$$\hat{cov}(f^q \mid D) = C = -(M-1)\frac{1}{\Sigma^{qq}} + \sum_{i=1}^{M} cov(f^q \mid D_i)^{-1} \tag{2.34}$$

2.3.2 Hierarchical Mixture of Experts

The Mixture-of-Experts (MoEs) are a well-know model that differ from the Mixture Models for the dependence of the mixture weights on the input. In fact, given an input vector $\mathbf{x}$ and an output y, being it either discrete (e.g. for classification) or continuous (e.g. for regression), the general formulation is [17]:

$$p(y \mid \mathbf{x}, \mathbf{W}, \mathbf{U}) = \sum_{h=1}^{H} p(y \mid \mathbf{x}, \mathbf{w}_h) p(h \mid \mathbf{x}, \mathbf{U}) \tag{2.35}$$

where H is the number of experts, $\mathbf{w}_h$ is the set of parameters for expert h (with $\mathbf{W} = \{\mathbf{w}_1, \ldots, \mathbf{w}_H\}$) and $\mathbf{U} = \{\mathbf{u}_1, \ldots, \mathbf{u}_H\}$ are the gating weights for each component. Every submodel is an "expert" of a certain portion of the input space and the *gating function*

$p(h \mid \mathbf{x}, \mathbf{U})$ usually takes the form of a softmax function:

$$p(h \mid \mathbf{x}, \mathbf{U}) = \frac{e^{\mathbf{u}_h^T \mathbf{x}}}{\sum_j e^{\mathbf{u}_j^T \mathbf{x}}} \tag{2.36}$$

Therefore, the system is able to determine whether the model h is suitable for a specific input vector through the weight vector $\mathbf{u}_h$.

The MoEs can be applied to any distribution and the experts are often chosen to be GPs, especially for regression [18]. When each expert is a MoE itself, it produces the so-called Hierarchical MoE (HME), first proposed in [19]. This model can be though of as a particular decision tree, since it involves the recursive partition of the space and the application of an expert to each region. GP-based HME (HGP) have recently been used to overcome the issue of scalability for large datasets that affects GPs applications. For eaxample, a two-layer hierarchy have been proposed in [20], which we briefly describe here.

The input space $\boldsymbol{X}$ is partitioned into Q non-overlapping regions, such that $\boldsymbol{X} = X_1 \cup \cdots \cup X_Q$ (with $|X_j| = N_j$), and a GP is assigned to each of them. The upper-level components of the HME are involved to roughly shape the data, whereas the individual GPs in each leaf precisely model the data points in that particular partition of the space.

Let $\boldsymbol{C} = \{\boldsymbol{c}_j\}_{j=1}^{Q}$ be the set of the means of every partition, we can use the functions g and f_j to model respectively the upper level and the j-th partition in the lower level. The GP prior on these latent functions would be:

$$f_j(\mathbf{x}) \mid g \sim \mathcal{GP}(g(\boldsymbol{c}_j), k_j(\mathbf{x}, \mathbf{x}')) \tag{2.37}$$

$$g(\boldsymbol{c}) \sim \mathcal{GP}(0, k_g(\boldsymbol{c}, \boldsymbol{c}')) \tag{2.38}$$

where $k_j(\mathbf{x}, \mathbf{x}')$ and $k_g(\boldsymbol{c}, \boldsymbol{c}')$ are the covariance functions defined on each subset of data points and on the partitions mean respectively.

The problem formulation is then similar to the one expressed in Section 2.2.3, although the dependence of one latent function on the other needs now to be addressed. Let us define $\boldsymbol{g} = [g_1, \ldots, g_Q]^T$, with $g_j = g(\boldsymbol{c}_j)$, and $\boldsymbol{f} = [\boldsymbol{f}_1^T, \ldots, \boldsymbol{f}_Q^T]^T$, with $\boldsymbol{f}_j = [f_1^j, \ldots, f_{N_j}^j]^T$ and $f_i^j = f_j(\mathbf{x}_i)$. The conditional distribution is:

$$p(\boldsymbol{f} \mid \boldsymbol{g}, \boldsymbol{X}) = \mathcal{N}(\boldsymbol{f} \mid \boldsymbol{H}\boldsymbol{g}, \boldsymbol{D}) \tag{2.39}$$

where $\boldsymbol{H}$ is a $N \times Q$ binary matrix indicating in $[\boldsymbol{H}]_{ij}$ whether point $\mathbf{x}_i$ belongs to partition j, and $[\boldsymbol{D}_j]_{lm} = k_j(\mathbf{x}_l^j, \mathbf{x}_m^j)$. The GP priors over $\boldsymbol{g}$ and $\boldsymbol{f}$ are Gaussian distributed and, in this case, are equal to:

$$p(\boldsymbol{g} \mid \boldsymbol{C}) = \mathcal{N}(\boldsymbol{g} \mid 0, \boldsymbol{K}_g) \tag{2.40}$$

$$p(\boldsymbol{f} \mid \boldsymbol{X}, \boldsymbol{C}) = \mathcal{N}(\boldsymbol{f} \mid 0, \boldsymbol{K}_f) \tag{2.41}$$

where $[\boldsymbol{K}_g]_{ij} = k_g(\boldsymbol{c}_i, \boldsymbol{c}_j)$ and $\boldsymbol{K}_f = \boldsymbol{H}\boldsymbol{K}_g\boldsymbol{H}^T + \boldsymbol{D}$.

Then, given a prediction $y_i^j = f_j(\mathbf{x}_i^j) + \varepsilon_i^j$ for the point $\mathbf{x}_i$ in the j-th partition, with $\varepsilon_i^j \sim \mathcal{N}(0, \sigma_i^2)$, the likelihood of the complete set of targets given $\boldsymbol{f}$ is:

$$p(\mathbf{y} \mid \boldsymbol{f}) = \mathcal{N}(\mathbf{y} \mid \boldsymbol{f}, \boldsymbol{\Lambda}) \tag{2.42}$$

where $\boldsymbol{\Lambda} = \text{diag}(\boldsymbol{\lambda}_1, \ldots, \boldsymbol{\lambda}_Q)$, and $\boldsymbol{\lambda}_j = [\sigma_j^2, \ldots, \sigma_j^2]$ is a N_j-dimensional row vector. We can now compute the posterior distribution of the lower-level values, derived from equations (2.41) and (2.42):

$$p(\boldsymbol{f} \mid \mathbf{y}, \boldsymbol{X}, \boldsymbol{C}) = \mathcal{N}(\boldsymbol{f} \mid \boldsymbol{\Sigma}_f \boldsymbol{\Lambda}^{-1}\mathbf{y}, \boldsymbol{\Sigma}_f) \tag{2.43}$$

where

$$\begin{aligned} \boldsymbol{\Sigma}_f &= \boldsymbol{K}_f \boldsymbol{\Sigma}_N^{-1} \boldsymbol{\Lambda} \\ \boldsymbol{\Sigma}_N &= \boldsymbol{H}\boldsymbol{K}_f^{-1}\boldsymbol{H}^T + \tilde{\boldsymbol{D}} \\ \tilde{\boldsymbol{D}} &= \boldsymbol{D} + \boldsymbol{\Lambda} \end{aligned}$$

The likelihood given $\boldsymbol{g}$ is:

$$\begin{aligned} p(\mathbf{y} \mid \boldsymbol{g}, \boldsymbol{X}) &= \int p(\mathbf{y} \mid \boldsymbol{f}) p(\boldsymbol{f} \mid \boldsymbol{X}, \boldsymbol{g}) d\boldsymbol{f} \\ &= \mathcal{N}(\mathbf{y} \mid \boldsymbol{H}\boldsymbol{g}, \tilde{\boldsymbol{D}}) \end{aligned} \tag{2.44}$$

Therefore, from equations (2.40) and (2.42), the posterior over $\boldsymbol{g}$ results to be:

$$p(\boldsymbol{g} \mid \mathbf{y}, \boldsymbol{C}) = \mathcal{N}(\boldsymbol{g} \mid \tilde{\boldsymbol{\mu}}, \boldsymbol{A}^{-1}) \tag{2.45}$$

where

$$\begin{aligned} \tilde{\boldsymbol{\mu}} &= \boldsymbol{A}^{-1}\boldsymbol{H}^T\tilde{\boldsymbol{D}}^{-1}\mathbf{y} \\ \boldsymbol{A} &= \boldsymbol{K}_g^{-1} + \boldsymbol{H}^T\tilde{\boldsymbol{D}}\boldsymbol{H} \end{aligned}$$

Methods

3.1 Hierarchical Bayesian Committee Machine

3.1.1 General approach

The BCM represents one of the latest approaches proposed in the literature to provide a solution for the large training dataset necessary in multi-domain problems. Specifically, it allows us to combine the estimates from different regions of the space and weight them according to the reliability of the committee member they are predicted by. In this way, we can overtake the eventually small amount of data available for certain domains and balance their predictions with those of the more occurring ones. Similarly, using GPs to model our problem presents the advantages of Bayesian non-parametric approaches. In particular, their ability to infer on unknown portions of the input space guides the exploration towards those regions that are more uncertain. This guarantees to learn the optimal policy by efficiently using the available data and balancing the exploitation and exploration of the belief state.

The formulation of the BCM described in Section 2.3.1 is easily applicable to the SDS problem, as also formulated in [15]. Specifically, we consider the function Q as our predictive distribution for GP regression, with a prior $Q(\mathbf{b},a) \sim \mathcal{GP}(0,k((\mathbf{b},a),(\mathbf{b},a)))$. In this case, the estimate of the Q-function for the i-th single domain has mean $\bar{Q}_i(\mathbf{b},a)$ and covariance $\Sigma_i^Q(\mathbf{b},a)$. Therefore, the combined estimate is calculated as:

$$\bar{Q}(\mathbf{b},a) = \Sigma^Q(\mathbf{b},a) \sum_{i=1}^{M} \Sigma_i^Q(\mathbf{b},a)^{-1} \bar{Q}_i(\mathbf{b},a) \tag{3.46}$$

$$\Sigma^Q(\mathbf{b},a)^{-1} = -(M-1)\frac{1}{k((\mathbf{b},a),(\mathbf{b},a))} + \sum_{i=1}^{M} \Sigma_i^Q(\mathbf{b},a)^{-1} \tag{3.47}$$

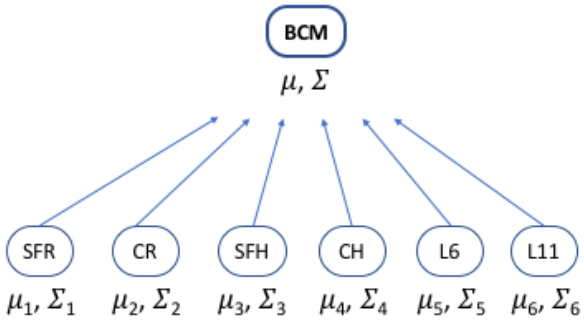


Figure 3.7. Architecture of the BCM for a set of six domains. We indicate $\bar{Q}_i(\mathbf{b}, a)$ with μ_i and $\Sigma_i^{\bar{Q}}(\mathbf{b}, a)$ with Σ_i.

As evident from the expression, the predictions from different estimators are weighted by the inverse of their covariance function. In other words, if any of the committee member is uncertain about the prediction, then its contribution would be automatically considered less relevant. An example of the BCM construction is illustrated in Figure 3.7, that shows the resulting architecture for a set of six domains selected from the ontology.

Given the list of domains available in the *cued-pydial* database, we notice that some of them may be thought of as related, according to different aspects. For example, we expect that the CR and CH instances might share some of the values in the database, that could be abstracted to form a "general" knowledge. Therefore, the idea we want to examine is the possibility of a generalisation among the domains, such that "similar" ones may contribute to each other's prediction.

For this purpose, we apply the HGP method defined in Section 2.3.2 to build a hierarchy of domains, in which the contribution from those that are related can be combined together. Jointly with the BCM model, we construct a hierarchical BCM (HBCM), by introducing an intermediate level of fictional domains (*parents*), each of them acting as BCMs. Then, these parent domains are combined as members of an upper-level BCM. In particular, we follow the same approach as in [20] and we consider a domain-specific policy at the leaves level, whereas the parent-level policies are assumed to provide a generalised policy by weighting the influence of their own children.

As previously mentioned, we exploit GPs to model the policy optimisation task. Thus, each of the leaves and parent domains in our architecture train a GP. In this case, both the parents and the children domains cover the role of a committee member. Thus, we simplify the formulation presented in Section 2.3.2 for the HGPs, by applying the BCM equations to the lower and upper levels. Consequently, the j-th parent mean and covariance predictions are

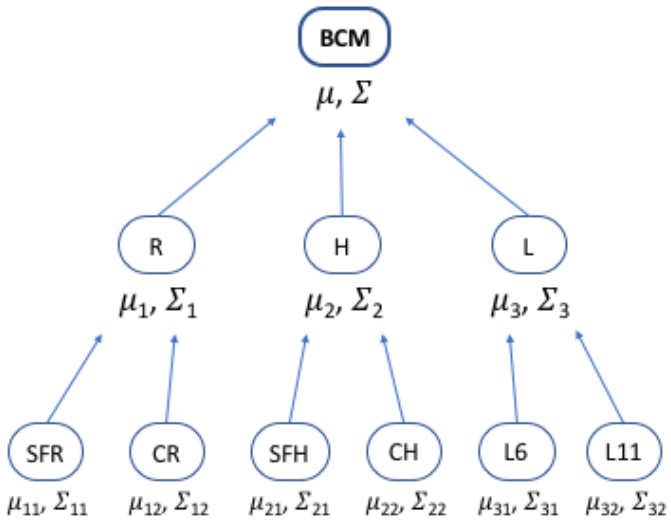


Figure 3.8. Architecture of the HBCM for a set of six domains. Here, we choose as parent domains Restaurants (R), Hotels (H) and Laptops (L), but other structures are possible. We indicate $\bar{Q}_{ji}(\mathbf{b},a)$ with μ_{ji} and $\Sigma^{Q}_{ji}(\mathbf{b},a)$ with Σ_{ji}.

formulated as:

$$\bar{Q}_j(\mathbf{b},a) = \Sigma^{Q}_{j}(\mathbf{b},a)\sum_{i=1}^{N}\Sigma^{Q}_{ji}(\mathbf{b},a)^{-1}\bar{Q}_{ji}(\mathbf{b},a) \tag{3.48}$$

$$\Sigma^{Q}_{j}(\mathbf{b},a)^{-1} = -(N-1)\frac{1}{k((\mathbf{b},a),(\mathbf{b},a))} + \sum_{i=1}^{N}\Sigma^{Q}_{ji}(\mathbf{b},a)^{-1} \tag{3.49}$$

where N is the number of children for the j-th parent node, and $\Sigma^{Q}_{ji}(\mathbf{b},a)$ and $\bar{Q}_{ji}(\mathbf{b},a)$ are the mean and covariance of its i-th child respectively.
The final prediction is computed as in the classic BCM for GPs, formulated in equation (3.47), by combining the predictions from the domains in the intermediate level. In this case, we indicate with M the number of parents in the architecture.

As an example, Figure 3.8 illustrates the same set of domains selected in Figure 3.7 and organised as a HBCM.

3.1.2 Automatic model selection

Given a set of domains, different ways of grouping them are possible, depending on the relation we wish to highlight between them. However, besides the possible hand-crafted

hierarchies, we might want to reduce the human intervention in the model setup. For this purpose, an automatic procedure for selecting an appropriate architecture can be implemented.

One option would be to perform a "greedy" approach to cluster the domains. For instance, we could start from a random domain and keep adding domains to the set as long as we obtain gains in the evaluation of the dialogue. This method will be explained more in detail in Section 5.2.1, since the running time would be prohibitive for the purposes of this work.

In this section, we rather describe a further approach, which might not be as accurate, though it is simpler to implement. In particular, we have seen how the entropy measure stresses the domains informative capability within the dialogue. Therefore, we exploit the vectors of normalised entropy and we group them through the K-means algorithm in order to find the best clustering, as described below.

K-means

The procedure we explain in this section makes use of the normalised entropy computation, formulated in equation (2.27). This metric allows an ordering of the slots in each domain by carrying the information about their informative power, and provides an indication on the complexity of the model. Here, we propose the K-means algorithm as method for the automatic model selection and we operate on the normalised entropy measure, computed for each domain. In other words, we build the vectors of the slots nomalised entropy and we use them as input feature vectors for the algorithm. In particular, we base this strategy on the assumption that similar domains may share latent information. In this way, we expect that grouping domains with similar values for the entropy vectors can lead to an overall simpler problem, which is easier for the BCM method to be handled.

Specifically, we construct the vectors as follows. For every domain d and for each of the two semantic classes separately (requestable and informable) we store in a vector the normalised entropies $\eta(s)$ for each slot s. In other words, if the set of domains is $\mathcal{D}$, we will have $D = |\mathcal{D}|$ pairs of vectors: one will contain the normalised entropies of the requestable slots (req_d) and the other those of the informable slots (inf_d):

$$\begin{aligned} req_d &= [\ \ \eta_1^{req} \ \ \cdots \ \ \eta_{r_d}^{req} \ \ ] \\ inf_d &= [\ \ \eta_1^{inf} \ \ \cdots \ \ \eta_{i_d}^{inf} \ \ ] \end{aligned} \qquad \text{with } d = 1, \ldots, D \tag{3.50}$$

where η_i^c is the normalised entropy for slot i of class c, and we keep the same slots ordering within each vector as explained in Section 2.3. We indicate with r_d and i_d respectively the

number of requestable and informable slots for the d-th domain.
Then, fixing a semantic class c (e.g. $c = req$), we consider the vectors $\{c_d\}_{d=1}^{D}$ from every domain and we pad them with zeros ($\bar{c}_d$ is the padded vector), so that their length will be equal to the longest vector's one. For instance, given a set of three domains, the padding for semantic class c would be:

$$\begin{array}{lll} \bar{c}_1 = [& \eta_1^c \ \cdots \ \eta_{r_1}^c \ 0 \ 0 &] \\ \bar{c}_2 = [& \eta_1^c \ \cdots \ \cdots \ \cdots \ \eta_{r_2}^c &] \\ \bar{c}_3 = [& \eta_1^c \ \cdots \ \cdots \ \eta_{r_3}^c \ 0 &] \end{array} \qquad \text{with } r_1 < r_3 < r_2 \tag{3.51}$$

Finally, for every domain d we concatenate the padded requestable ($\bar{req}_d$) and informable ($\bar{inf}_d$) vectors to build a single vector v_d:

$$v_d = \left[\bar{req}_d \quad \bar{inf}_d \right], \quad \text{with } d = 1, \ldots, D \tag{3.52}$$

We address now the problem of clustering the feature vectors v_d we built, in order to consequently assign a parent domain to each subset. For this purpose, we use the K-means algorithm, that aims to partition the input vectors into K clusters. A cluster is a set of points for which the distances are small compared to those of points outside the cluster, and to which we can then associate its mean $\boldsymbol{\mu}_k$. We use the binary variable z_{nk} to describe the assignment of point $\mathbf{x}_n$ to cluster k.

The goal is to find an assignment of data points to clusters and a set of mean vectors such that the sum of squared distances of each data point to its closest vector $\boldsymbol{\mu}_k$ is minimised [21]. The algorithm iterates between two steps, and can be thought of as a particular version of the Expectation-Maximisation (EM) algorithm in which we perform hard assignment to clusters [22]. In the first step, we assign the data point $\mathbf{x}_n$ to the cluster k if the k-th mean vector is the closest to the point:

$$z_{nk} = \begin{cases} 1 & \text{if } \ k = \text{argmin}_j \, || \, \mathbf{x}_n - \boldsymbol{\mu}_j \, ||^2 \\ 0 & \text{otherwise} \end{cases} \tag{3.53}$$

On the other hand, the second step involves the update of the mean vector for each cluster:

$$\boldsymbol{\mu}_k = \frac{\sum_n z_{nk} \mathbf{x}_n}{\sum_n z_{nk}} \tag{3.54}$$

The particular task we implement the algorithm for requires two considerations. Specifically, we want to avoid the extreme cases of the clustering, that is either having singletons ($K = D$) or a single cluster containing all the domains ($K = 1$). Moreover, since the number of clusters is a variable itself in our implementation, we operate a grid search to select the best value for K. We use the Bayesian Information Criterion (BIC) as model selection criterion [22]:

$$BIC = \log p(\mathcal{D} \mid \hat{\boldsymbol{\theta}}) - \frac{\text{dof}(\hat{\boldsymbol{\theta}})}{2} \log N \tag{3.55}$$

with

$$\log p(\mathcal{D} \mid \hat{\boldsymbol{\theta}}) = -\frac{N}{2} \log(2\pi\hat{\sigma}^2) - \frac{N}{2} \tag{3.56}$$

where $\hat{\boldsymbol{\theta}}$ is the Maximum Likelihood Estimate (MLE) for the model, $\text{dof}(\hat{\boldsymbol{\theta}})$ is its number of degrees of freedom, N is the number of data points and $\hat{\sigma}$ is the variance of the input features. The best model architecture is then selected by finding the clustering with the highest BIC score.

Experiments

4.1 Experimental setup

In this chapter, we describe the configurations chosen for the experiments we carried out and the correspondent results, which are given with 95% confidence intervals. The standard setup for all the experiments is defined below.

We make use of the user simulator provided within the *cued-pydial*, for which the semantic error rate is set to 15% both in training and testing steps. Specifically, during training we use 200 dialogues for each of the domains selected (unless differently specified), whereas we evaluate the model on 1000 dialogues in every case. Every dialogue is limited to a maximum of 30 turns and is constrained to deal with only one domain. We assign a negative reward (-1) for each turn of the dialogue and a $+20$ if it is successful.

The belief state tracker is implemented as focus tracker, whereas the GPSARSA algorithm is used to optimise the policy. In particular, for the latter, the sparsification threshold used to fix the size of the representative points dictionary is $\nu = 0.01$. In addition, among the several possible kernel functions [9], we choose a linear kernel and the Kronecker delta for the covariance function, respectively as belief and action kernels:

$$k_{\mathcal{B}}(\mathbf{b}, \mathbf{b}') = \sum_i \langle \mathbf{b}_i, \mathbf{b}'_i \rangle \tag{4.57}$$

$$k_{\mathcal{A}}(a, a') = \delta_a(a') = \begin{cases} 1 & \text{iff } \ a = a' \\ 0 & \text{otherwise} \end{cases} \tag{4.58}$$

In this way, we can handle both the continuity of the belief state space and the discrete actions available.

In the next sections, we explain the experiments performed and the outcomes obtained. Further and more detailed results are reported in the Appendix, when indicated.

4.2 In-domain and BCM comparison

In this section, we follow the procedure presented in [15] and we replicate some of the experiments described, though the results vary slightly due to the different framework used. Specifically, we pick a subset of domains from the ontology and we compare the in-domain trained policy with the performance of the BCM, according to the following set up. The in-domain policy is obviously trained and evaluated on the same single domain, chosen among L6, SFR and L11. On the other hand, the BCM uses all the domains selected to jointly train the model, whereas the evaluation is done on a single domain at time. The configurations in this case are:

- Train on {SFR, SFH, L6} and test on L6.
- Train on {SFR, SFH, L11} and test on L11.
- Train on {SFR, SFH, L11} and test on SFR.

Each of these setups is examined both in a small-dataset case (250 training dialogues per domain) and when more data is available (2500 training dialogues per domain).

We report the results in Table 4.1. For the BCM case we refer to the committees above by only indicating the tested domain. The figures confirm the dramatic improvement of the policy optimisation outcome provided by the introduction of the BCM. In fact, the gains in performance are evident in every experiment, although the increase varies depending on the number of dialogues used for training. When fewer dialogues are used, for example, the

Table 4.1. Comparison of the in-domain and BCM performances when training data with different sizes are used. The number of training dialogues for each domain is indicated, as well as the tested domain (third column).

Training dialogues	Method	Domain	Reward	Success	Turns
250	in-domain	L6	3.57 ± 0.74	60.97 ± 3.02	8.63 ± 0.33
		L11	3.82 ± 0.76	66.17 ± 2.91	9.42 ± 0.38
		SFR	5.52 ± 0.71	69.67 ± 2.78	8.42 ± 0.31
	BCM	L6	9.66 ± 1.53	88.27 ± 1.46	7.99 ± 1.24
		L11	4.61 ± 2.15	83.97 ± 2.92	12.18 ± 1.57
		SFR	8.41 ± 1.21	86.37 ± 3.40	8.86 ± 0.60
2500	in-domain	L6	11.58 ± 0.39	91.62 ± 1.50	6.74 ± 0.18
		L11	10.00 ± 0.55	86.23 ± 2.13	7.25 ± 0.25
		SFR	10.61 ± 0.50	89.10 ± 1.93	7.21 ± 0.21
	BCM	L6	12.04 ± 0.81	94.83 ± 3.24	6.93 ± 0.16
		L11	10.23 ± 0.16	91.47 ± 0.80	8.06 ± 0.12
		SFR	11.45 ± 0.35	92.60 ± 0.85	7.07 ± 0.18

success rate is improved of more than 15%. Similarly, the reward grows of up to 6 points (e.g. see L6 for 250 training dialogues), and presents a more notable gap in the small dataset case. In general, when only a small number of training dialogues is available the system is more subject to errors and the final reward decreases with respect to the 2500-dialogues case.

In addition, we notice that the L11 performance is low compared to the other domains in almost every case. This is due to the high number of informable and, in particular, requestable slots that are available for this domain. As a consequence, the number of variables to be controlled by the system for achieving the user's goal increases, which results in a more complex task. This outcome is even more evident when the training dataset is small, since the model would need more data to optimise the policy for a hard problem.

In this regard, a higher number of dialogues appears to guarantee a general improvement of the performances even when the in-domain training is performed, providing an accuracy that is close to or exceeds 90% for almost every domain chosen. On the other hand, the largest gain in performance is obtained when the BCM is applied to small-dataset problems, which especially stresses the usefulness of this method. In fact, an overall increasing trend is observed when the BCM strategy is preferred.

4.3 BCM and HBCM comparison

In this section, we compare the BCM results to different hierarchical architectures, examining the performance as the number of domains varies. For this purpose, we consider the following two configurations.

4.3.1 Small ontology

The domains selected for this experiment are SFR, CR, SFH, CH, L6 and L11. We fix the number of training dialogues to 200 for each domain and we run the evaluation using the BCM and the HBCM. Specifically, the parent domains in the latter are manually chosen to be Restaurants, Hotels and Laptops, so that the three clusters are:

$$\begin{aligned} \text{Restaurants (R)} &= \{\text{SFR, CR}\} \\ \text{Hotels (H)} &= \{\text{SFH, CH}\} \\ \text{Laptops (L)} &= \{\text{L6, L11}\} \end{aligned} \tag{4.59}$$

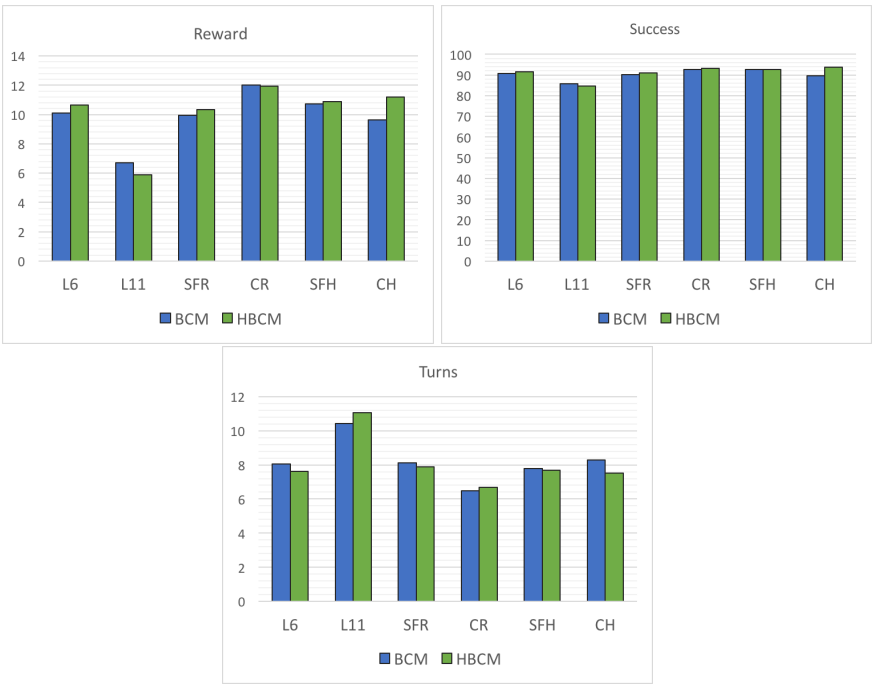


Figure 4.9. Performance of the HBCM compared to the BCM on the set of six domains selected in the small-ontology case.

From the results illustrated in Figure 4.9 (see also Table A.1), we notice that the introduction of the hierarchical approach does not provide any significant gain overall. In this case, in fact, for almost all the domains the success rate is comparable for both the strategy used. A similar trend is observable concerning the reward, although the figures show an evident increase (1.5 points approximately) when the CH domain is evaluated. This may find its reason in the smaller number of turns that are necessary to achieve the goal.
Finally, the same poor performance is obtained for L11, meaning the HBCM was not able to overcome the complexity of that particular domain in this setup.

4.3.2 Large ontology

Following the previous experiments, here we want to examine the performance of the BCM and HBCM when the task is scaled to a larger number of domains. In this case, we consider the full set of domains listed in Section 2.1.2. Two different hand-crafted hierarchies are compared, namely HBCM-*area* and HBCM-*topic*. In particular, we define the clusters as follows:

- HBCM-area:

$$\begin{aligned} \text{SanFrancisco (SF)} &= \{\text{SFR, SFH}\} \\ \text{Cambridge (C)} &= \{\text{CR, CH, CS, CA, CT}\} \\ \text{Electronics (E)} &= \{\text{L6, L11, TSBp, TV}\} \end{aligned} \tag{4.60}$$

- HBCM-topic:

$$\begin{aligned} \text{Restaurants (R)} &= \{\text{SFR, CR}\} \\ \text{Hotels (H)} &= \{\text{SFH, CH}\} \\ \text{Shops (S)} &= \{\text{CS}\} \\ \text{Attractions (A)} &= \{\text{CA}\} \\ \text{Transport (T)} &= \{\text{CT}\} \\ \text{Electronics (E)} &= \{\text{L6, L11, TSBp, TV}\} \end{aligned} \tag{4.61}$$

The main observation that results from the performance analysis is the very poor outcome provided by the hierarchy HBCM-topic (see Figure 4.10 and Table A.2 for details). This may be caused by the smaller cardinality of some of the subsets defined in (4.61), compared to those in (4.60). In other words, branches of the hierarchy such as S or A represent singletons and the policy can rely on one domain only. Therefore, if the prediction is uncertain, no other contribution is available to balance or improve it at the lower level of the architecture.

On the other hand, when the HBCM-area is used, the parent domains correspond to clusterings that contain a larger number of domains. This appears to be an aid in achieving a better policy within the branches and, consequently, a better policy overall. In fact, as reported in Table A.2, the performance is increased with respect to the BCM of a small but approximately constant value for almost all the domains, both in terms of success rate and reward.
It is interesting to notice that, in the HBCM-area case, modest gains on L11 are also achieved, with respect to the BCM results. This means that the domains in E are able to improve the predictions on the L11 domain, as well. Conversely, although the composition of the parent domain E is the same, in the HBCM-topic configuration the performance for L11 dramatically decreases. As mentioned before, this may find its reason in the combination of the E predictions with the poor ones provided by the singleton parents in the latter hierarchy.

As a consequence, model selection has been shown to be a fundamental issue, not only to reduce the need of human intervention, but also because not every architecture may be suitable for the task resolution. The latter problem is also reported in the literature for similar

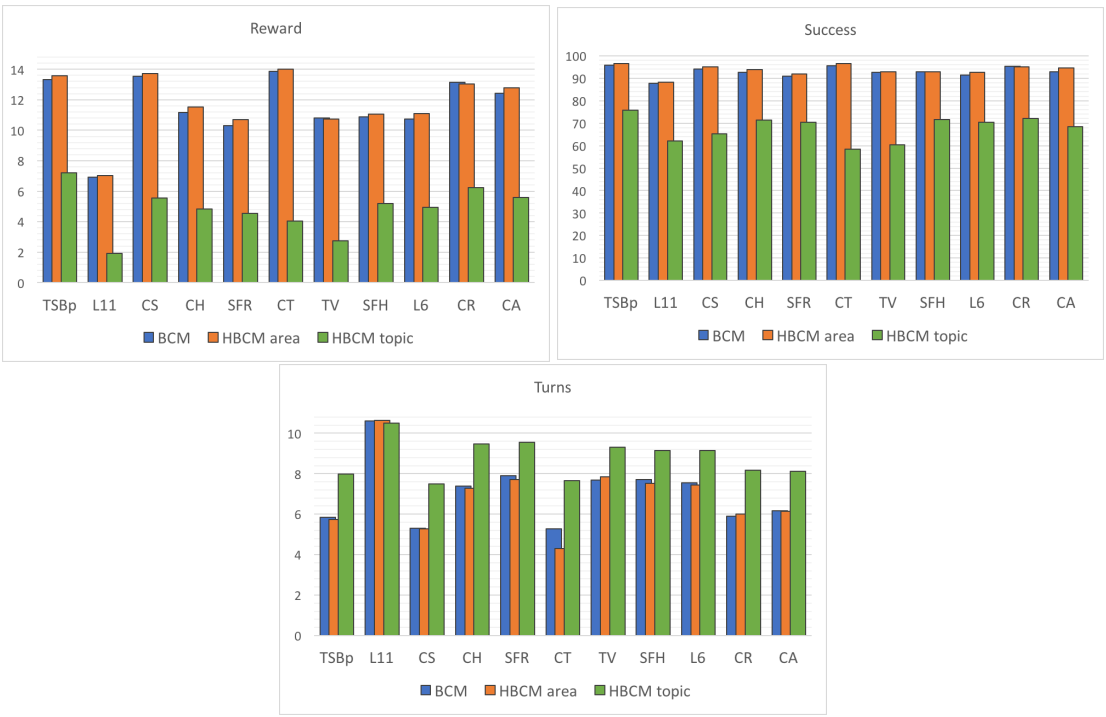


Figure 4.10. Performance of the BCM on the large-ontology set of domains, compared to the two hand-crafted hierarchies.

hierarchical tasks [20]. Finally, comparing to the small-ontology outcomes, a larger ontology appears to generalise better on the single domains and guarantees an increase of at least 1% on the majority of the domains considered (compare Tables A.1 and A.2).

4.4 Automatic model selection

Given the observations derived from the previous section, we examine the possibility to build the hierarchy automatically to overcome the model selection problem. In particular, we use the K-means algorithm on the normalised entropy vectors of the domains, as described in Section 3.1.2, and we tune the number of clusters k by maximising the BIC score. Specifically, we carry out this procedure in two different initial settings:

- We build the vectors v_d basing on the requestable slots only.
- We use both the requestable and the informable slots to build the normalised entropy vectors v_d.

After a brief evaluation over 50 dialogues, the latter setup result to be more robust, meaning that the contribution of the informable slots is still relevant and represents an aid to the overall performance. This is still valid although the requestable slots are the main source to the system to discern among the suitable entities and accomplish the user's goal. Consequently, we use both the semantic classes for the actual experiments in this section.

The hierarchy resulting from the K-means clustering (HBCM-*auto*) is given below, where we assign a fictional name P_i to the parent domains since we are not interested in their semantic meaning:

$$\begin{aligned} P_1 &= \{\text{SFR, SFH, CR, L11}\} \\ P_2 &= \{\text{CH, CA, TV}\} \\ P_3 &= \{\text{CS, CT, L6, TSBp}\} \end{aligned} \tag{4.62}$$

The policy is then trained basing on this architecture and evaluated on every single domain. In Table 4.2 we present the results in terms of final reward, percentage of success and number of turns, averaged over all the domains. The performance for each domain singularly is reported in detail in Table A.3.

The comparison of the averaged results shows what has already been proved in the previous experiments, that is the HBCM may guarantee gains in performance with respect to the BCM. However, it is clear that the general outcome strictly depends on the chosen architecture. In fact, while the HBCM-topic appears to be poorly performing, the HBCM-area guarantees a higher rate of success of almost 1 point comparing to the classical BCM.

On the other hand, the method used to automatically select the clustering presents a wide margin for improvement, since both the final average reward and accuracy are not satisfactorily comparable to the BCM performance. In fact, as illustrated in Figure 4.11, although in rare cases the rewards and success rate are barely higher for the HBCM-auto, the general trend

Table 4.2. Performance of the models examined in this work, averaged over all the domains. We compare the BCM, the two hand-crafted hierarchies (HBCM-topic, HBCM-area) and the model automatically built (HBCM-auto).

Method	Reward	Success	Turns
BCM	11.56 ± 1.19	92.92 ± 1.40	7.03 ± 0.92
HBCM topic	4.80 ± 0.89	67.86 ± 3.29	8.77 ± 0.56
HBCM area	11.75 ± 1.19	93.70 ± 1.40	6.90 ± 0.99
HBCM auto	10.63 ± 1.52	88.27 ± 4.60	7.02 ± 0.82

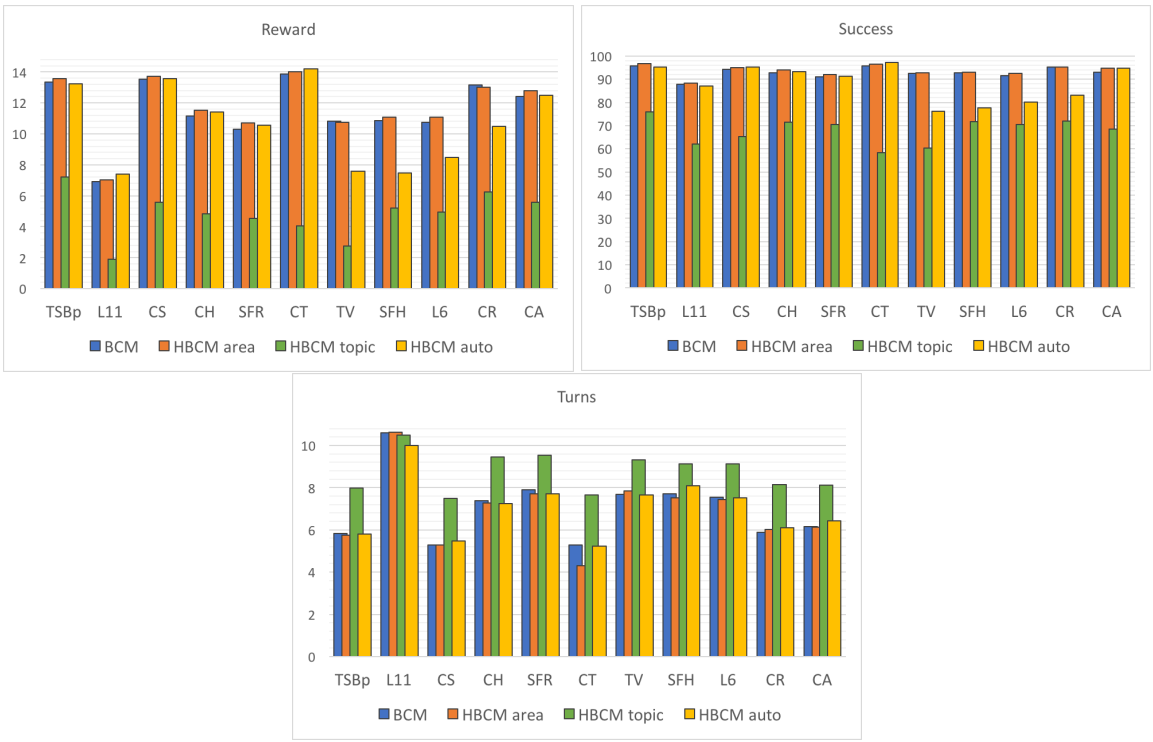


Figure 4.11. Performance of the HBCM-auto on every domain, compared to the BCM and the hand-crafted models performance from the previous sections.

is of a similar performance, with dramatic drops in the outcomes for certain domains (e.g. see TV, SFH, L6, CR). This may be due to a scarce capability of gathering the underlying information in common among domains in the same subset of the hierarchy. Similarly, the prediction on one domain might be affected by the high uncertainty on a "sibling" in the same branch. One of the reasons could be that the application of the K-means algorithm on the normalised entropy vectors might not be suitable for such a small number of domains. Therefore, we propose other possible alternatives for automatic model selection in Section 5.2.

4.5 Domains contribution

Finally, we want to examine the correlation among domains, that is how useful each committee is. We experiment on the small-ontology subset of six domains (SFR, CR, SFH, CH, L6, L11), maintaining the same clustering as in (4.59). In particular, we divide the experiments as follows:

- Train on R only.

Table 4.3. Contribution of different domain subsets combined together.

Parents Committee	Domain	Reward	Success	Turns
R	SFR	4.15 ± 1.06	80.07 ± 1.99	11.86 ± 0.96
	CR	10.33 ± 1.21	90.22 ± 3.41	7.72 ± 0.62
R + H	SFR	9.40 ± 0.59	89.33 ± 0.58	8.47 ± 0.57
	CR	11.98 ± 1.20	93.47 ± 2.04	6.71 ± 0.96
R + L	SFR	7.90 ± 1.07	85.37 ± 3.56	9.18 ± 0.96
	CR	11.28 ± 0.66	90.97 ± 3.04	6.91 ± 0.24
R + H + L	SFR	10.32 ± 0.26	91.00 ± 1.03	7.88 ± 0.09
	CR	11.93 ± 0.46	93.08 ± 0.93	6.69 ± 0.47

- Train on {R, H}.
- Train on {R, L}.
- Train on {R, H, L} (same as the six-domains case in Section 4.3.1).

In all these cases, the model is evaluated on SFR and CR only, in order to be able to compare the outcomes.

The results reported in Table 4.3 highlight that the choice of the domains influences the performance during the evaluation. For instance, when both R and H are used for training, the reward increases of at least 1.5 points, compared to the case in which only R is used. Similarly, the success rate approximately soars of 9% and 3% for SFR and CR respectively. On the contrary, the combination of R with the Laptops group of domains still improves the policy, but the final performance is significantly lower than in the {R, H} case. However, we observe that, besides the small contribution in the {R, L} case, the addition of the L subset positively affects the overall performance when all the domains selected are used. As an example, the evaluation on SFR presents gains in both the reward of the dialogue and the rate of success, whereas it maintains approximately the same outcome for CR.

Given the definition from the ontology of the domains in the Restaurants and Hotels branches, the results suggest that not all the domains are useful in the same way, and that "similar" domains provide an aid to the general policy. In this case, for instance, the lower rewards and accuracies indicates a lack of shared information between the R and L instances. In addition, complex domains such as L11, in which the number of slots is large, might contribute to a decrease in the overall performance. Therefore, the results stress again the importance of the correct combination of domains, in order to optimise and balance the uncertainty of the predictions. Specifically, they validate the clustering approach we carry out in Section 4.4

and explain why the HBCM-auto is better performing than the HBCM-topic. The singleton subsets in the latter, in fact, suffer from the lack of a sibling domain to balance the policy learning. A similar situation is represented here when the R parent domain is used alone rather than combined with either H or L, in which case the lowest outcomes are achieved.

Conclusion

5.1 Discussion

In a SDS context, the policy optimisation task is one of the main problem to be dealt with in order to achieve a satisfactory performance of the system. In particular, the errors derived from the recognition of the user's intention and the small training data that is usually available are two of the main issues that we need to solve. Specifically, the latter is even more evident when a multi-domain model is used and managing multiple policies with a limited amount of data is fundamental.

We have presented a method for trying to exploit the similarities between domains by organising them in a hierarchical fashion. We prefer the use of GPs to model the dialogue decision rule to obtain a measure of uncertainty on our model predictions, which is intrinsically able to balance the exploration and exploitation of the belief state space. The underlying idea involves the clustering of the data into a (manually or automatically) defined number of subsets, each of which corresponds to a fictional parent domain. Every domain in the subset represents a Bayesian committee member and contributes to the parent prediction, as well as each parent is a Bayesian expert of the upper-level committee.

Overall, the HBCM provides a slight improvement in the performance with respect to the BCM, especially when we increase the number of domains considered. In other words, the hierarchical structure appears to increase the generalisation capability of the model for larger ontologies, which might be even more evident when further domains are included.

However, the gains are strongly dependent on the composition of the architecture used, which stresses the need for a suitable algorithm to solve the model selection issue. In this regard, the method implemented in this work resulted to be poor and not able to find the best performing clustering, probably because of the unsuitable input features chosen.

In general, the hierarchical nature of the model leaves open further investigations from multiple points of view. Unfortunately, deeper analyses could not be performed due to the lack of time, some of which are briefly described in the next section.

5.2 Future work

The performance of the model implemented in this work suggests that a variety of improvements that can be developed. First of all, further experiments could be performed to evaluate the possibility of scaling the method proposed to larger datasets. In addition, in the following sections we present other interesting analyses that could not be carried out due to the restricted time available.

5.2.1 Greedy architecture selection

Besides the method described in previous sections for automatically selecting a hierarchical model, several different approaches are possible. For instance, different metrics can be used to compare and select the best architecture, such as the Akaike Information Criterion (AIC) or the Minumum Description Length (MDL) [2]. Moreover, as illustrated in the experiments description, the K-means clustering does not appear to sufficiently stress the correlations between the domains, thus different methods could be applied.

In this section, we describe a "greedy" approach that may guarantee a more efficient implementation and a better performing system. Conversely to the K-means method, we will not use any auxiliary metric, but we directly evaluate on a fixed number of dialogues to have a measure of the "goodness" of the model. In this way, we could provide a more correct evaluation of the model itself because it is based on a short training step on the set of dialogues and the policy is evaluated directly. The following method avoids the need to rely on the external measure provided by the entropy vectors, as well.

First of all, we need to set up the clustering. For this procedure only, a training set of 50 dialogues for each domain can be used. We iterate the following steps as described below:

1. We randomly select a domain d_i, add it to the set $\mathcal{S}_k$ and evaluate the dialogues on the latter.

2. We randomly choose a further domain d_j and evaluate the model on $\mathcal{S}_k \cup d_j$.

[2] Refer to [22] for detail on these measures.

If the final reward is higher in the second case, then d_j is added to $\mathcal{S}_k$ and we repeat the procedure from step 2 with $\mathcal{S}_k = \{d_i, d_j\}$. Otherwise, a new set $\mathcal{S}_l$ is defined and the current chosen domain is assigned to it, so that $\mathcal{S}_k = \{d_i\}$ and $\mathcal{S}_l = \{d_j\}$. Then we repeat from step 1 considering the set $\mathcal{S}_l$. In other words, we add domains in one group (parent domain) while the reward gained from the dialogue evaluation improves, and we set up a new cluster otherwise.

The resulting architecture may depend on the order the domains are chosen. Thus, it is suggested to repeat this procedure with at least three different random starts and select the clustering that provides the best reward.

5.2.2 Parallelisation

The nature of the hierarchical model proposed suggests the possibility of parallelising the computations at the lower level. In fact, the data from one subset are independent of those in the other branches and their predictions are combined in each cluster independently.

The study carried out in [23] is based on the application of HMEs to large datasets by distributing the computations among the subsets of the hierarchy. Specifically, the author examines the case of GPs as experts and proposes the use of separate processes to perform simultaneous computations in a Python environment.

This approach could be easily applied to our model and would require minimal implementation. Basically, the mean and covariance predictions for the siblings in a branch can be combined independently of the other subsets. Therefore, the computations formulated in equation (3.48) can be performed in parallel for each parent domain. As a consequence, the overall computational cost of the policy optimisation could be reduced and contribute to the development of on-line applications of SDSs.

5.2.3 Avoiding sparse approximation

From the results reported in the previous chapter, it appears that the HBCM model outperforms the BCM especially when a larger ontology is used. However, despite the huge advantages that come with their usage for the policy formulation, GPs are not able to scale efficiently to a large input dataset. This results in a prohibitive computational cost of GP-based approaches when the multi-domain task involves a high number of domains.

Several approximated solutions have been proposed in the literature, such as the kernel span sparsification method we used for GPSARSA. Additionally to the parallelised operations mentioned in Section 5.2.2, in this section we suggest a further method that might reduce the computational cost. In particular, it could overcome the need for sparse approximations, allowing the application of the HBCM independently of the dimension of the ontology. The underlying idea is based on the model developed in [20]. Here, an approximated solution for HGPs is proposed, which allows the covariance matrix to be directly evaluated based on the data and its hierarchical structure, rather than selecting a set of inducing points. We briefly describe here the general idea and the possible application to SDSs problems.

The author suggests the exploitation of the GP-based hierarchy to discern two levels of learning. In the leaves level the whole set of data in each partition is used for modeling the cluster-specific GP, whereas the mean vectors of these clusters are used for the final prediction. Specifically, the latent variables that define the relationship between points in the same cluster are integrated out in the upper-level. As described in the paper, this results in a block covariance matrix in which diagonal blocks (referring to points in the same partition) are calculated exactly, while the off-diagonal elements are approximated thanks to the cluster-specific predictions.

In an SDS context, the input space is composed by the belief state ($\mathcal{B}$) and action ($\mathcal{A}$) spaces, and we consider it to be partitioned basing on the domains in the ontology. In other words, the subset corresponding to the domain d would contain the portions of $\mathcal{B}$ and $\mathcal{A}$ that relate to d. In contrast to the approach we applied, the author rejects the need of selecting and updating a set of representative points of the input space. We can apply this idea to our problem of the GPSARSA optimisation by partitioning the input space of each domain d in the ontology. Then, the mean $\bar{Q}_d(\mathbf{b}, a)$ and covariance $\Sigma_d^Q(\mathbf{b}, a)$ functions estimation for that domain are calculated based on the method proposed in the paper. These corresponds to the mean and covariance functions of the d-th committee member of the BCM. The application to the HBCM is straightforward.

This approach is based on the idea of dividing the complex task represented by the SDS policy optimisation of the single domains in order to handling smaller and simpler problems. The avoidability of the approximation is also considered by the author as a indication of robustness to overfitting, and therefore might improve the overall performance in a multi-domain task.

References

[1] Steve Young, Milica Gašić, Blaise Thomson, and Jason D Williams. Pomdp-based statistical spoken dialog systems: A review. *Proceedings of the IEEE*, 101(5):1160–1179, 2013.

[2] Matthew Henderson, Blaise Thomson, and Jason Williams. Dialog state tracking challenge 2 & 3, 2013.

[3] Jost Schatzmann, Blaise Thomson, Karl Weilhammer, Hui Ye, and Steve Young. Agenda-based user simulation for bootstrapping a pomdp dialogue system. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Companion Volume, Short Papers*, pages 149–152. Association for Computational Linguistics, 2007.

[4] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[5] Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1):99–134, 1998.

[6] Quentin JM Huys, Anthony Cruickshank, and Peggy Seriès. Reward-based learning, model-based and model-free. In *Encyclopedia of Computational Neuroscience*, pages 2634–2641. Springer, 2015.

[7] Milica Gasic and Steve Young. Gaussian processes for pomdp-based dialogue manager optimization. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 22(1):28–40, 2014.

[8] Carl Edward Rasmussen. Gaussian processes in machine learning. In *Advanced lectures on machine learning*, pages 63–71. Springer, 2004.

[9] Carl Edward Rasmussen. Gaussian processes for machine learning. MIT Press, 2006.

[10] Yaakov Engel, Shie Mannor, and Ron Meir. Reinforcement learning with gaussian processes. In *Proceedings of the 22nd international conference on Machine learning*, pages 201–208. ACM, 2005.

[11] Yaakov Engel. *Algorithms and representations for reinforcement learning*. Hebrew University of Jerusalem, 2005.

[12] Milica Gašić, Dongho Kim, Pirros Tsiakoulis, and Steve Young. Distributed dialogue policies for multi-domain statistical dialogue management. In *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, pages 5371–5375. IEEE, 2015.

[13] Volker Tresp. A bayesian committee machine. *Neural computation*, 12(11):2719–2741, 2000.

[14] M Gašić, N Mrkšić, Pei-hao Su, David Vandyke, Tsung-Hsien Wen, and Steve Young. Policy committee for adaptation in multi-domain spoken dialogue systems. In *Automatic Speech Recognition and Understanding (ASRU), 2015 IEEE Workshop on*, pages 806–812. IEEE, 2015.

[15] Milica Gašić, Nikola Mrkšić, Lina M Rojas-Barahona, Pei-Hao Su, Stefan Ultes, David Vandyke, Tsung-Hsien Wen, and Steve Young. Dialogue manager domain adaptation using gaussian process reinforcement learning. *Computer Speech & Language*, 2016.

[16] Volker Tresp. Committee machines, 2001.

[17] David Barber. *Bayesian reasoning and machine learning*. Cambridge University Press, 2012.

[18] Volker Tresp. Mixtures of gaussian processes. In *Advances in neural information processing systems*, pages 654–660, 2001.

[19] Michael I Jordan and Robert A Jacobs. Hierarchical mixtures of experts and the em algorithm. *Neural computation*, 6(2):181–214, 1994.

[20] Sunho Park and Seungjin Choi. Hierarchical gaussian process regression. In *Proceedings of 2nd Asian Conference on Machine Learning*, pages 95–110, 2010.

[21] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[22] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[23] Jun Wei Ng and Marc Peter Deisenroth. Hierarchical mixture-of-experts model for large-scale gaussian process regression. *arXiv preprint arXiv:1412.3078*, 2014.

Appendix A

	SFR	CR	SFH	CH	L6	L11	CS	CA	CT	TSBp	TV
requestable slots	pricerange allowedforkids near goodformeal food area	pricerange area food	dogsallowed pricerange near area acceptcreditcards hasinternet	pricerange area kind stars hasparking	batteryratings pricerange weightrange isforbusiness driverange family	batteryratings pricerange weightrange isforbusiness driverange family platform warranty utility processorclass sysmemory	area shopkind	area pricerange kind	transtype area	kind pricerange internet	screensizerange hdmi series usb eco pricerange
informable slots	postcode phone addr price	signature postcode description phone addr	postcode phone addr	booktime postcode price phone hasinternet booknumber bookdate bookname addr	drive dimension price	drive battery display design graphadaptor weight dimension processor price	addr phone postcode	price openhours addr postcode phone	addr phone postcode	weight dimension price hdmi	pixels accessories screensize cabinet audio power price

Figure A.1. Requestable and informable slots of the domains used. The slots are ordered basing on their normalised entropy within each semantic class.

Table A.1. BCM and HBCM performances on the small-ontology case (Section 4.3.1), involving a set of six domains.

	Domain	Reward	Success	Turns
BCM				
	L6	10.09 ± 0.28	90.73 ± 1.42	8.06 ± 0.53
	L11	6.72 ± 0.73	85.63 ± 2.24	10.41 ± 0.71
	SFR	9.93 ± 0.98	90.18 ± 1.13	8.11 ± 0.88
	CR	12.03 ± 0.63	92.63 ± 1.21	6.50 ± 0.56
	SFH	10.74 ± 0.29	92.68 ± 2.65	7.79 ± 0.57
	CH	9.63 ± 2.26	89.55 ± 3.80	8.29 ± 1.70
HBCM				
	L6	10.66 ± 0.23	91.43 ± 0.96	7.63 ± 0.08
	L11	5.87 ± 1.09	84.60 ± 2.48	11.05 ± 0.60
	SFR	10.32 ± 0.26	91.00 ± 1.03	7.88 ± 0.09
	CR	11.93 ± 0.46	93.08 ± 0.93	6.69 ± 0.47
	SFH	10.86 ± 0.32	92.75 ± 1.30	7.69 ± 0.21
	CH	11.21 ± 0.54	93.63 ± 1.24	7.52 ± 0.30

Table A.2. BCM and hand-crafted HBCMs performances on the large-ontology case (Section 4.3.2).

	Domain	Reward	Success	Turns
BCM				
	TSBp	13.33 ± 0.34	95.86 ± 1.11	5.84 ± 0.22
	L11	6.93 ± 0.57	87.72 ± 1.19	10.61 ± 0.34
	CS	13.54 ± 0.43	94.16 ± 2.05	5.30 ± 0.21
	CH	11.15 ± 0.04	92.72 ± 0.75	7.39 ± 0.17
	SFR	10.29 ± 0.25	90.92 ± 0.72	7.89 ± 0.12
	CT	13.86 ± 0.43	95.68 ± 1.41	5.28 ± 0.30
	TV	10.82 ± 0.42	92.56 ± 1.08	7.69 ± 0.34
	SFH	10.86 ± 0.35	92.82 ± 1.32	7.70 ± 0.25
	L6	10.74 ± 0.27	91.48 ± 0.86	7.56 ± 0.16
	CR	13.15 ± 0.17	95.22 ± 0.39	5.89 ± 0.19
	CA	12.43 ± 0.24	92.96 ± 0.7	6.16 ± 0.15
HBCM topic				
	TSBp	7.20 ± 0.55	75.88 ± 3.67	7.97 ± 0.25
	L11	1.91 ± 1.17	62.02 ± 4.53	10.50 ± 0.65
	CS	5.56 ± 0.93	65.30 ± 7.72	7.50 ± 0.63
	CH	4.82 ± 0.51	71.40 ± 3.44	9.46 ± 0.82
	SFR	4.55 ± 0.41	70.48 ± 4.98	9.55 ± 0.70
	CT	4.04 ± 1.05	58.44 ± 5.50	7.65 ± 1.14
	TV	2.74 ± 1.56	60.30 ± 5.22	9.32 ± 0.68
	SFH	5.20 ± 0.54	71.70 ± 4.54	9.14 ± 0.51
	L6	4.95 ± 1.01	70.44 ± 4.05	9.14 ± 0.63
	CR	6.24 ± 0.72	72.00 ± 3.90	8.16 ± 0.55
	CA	5.58 ± 1.02	68.54 ± 5.67	8.12 ± 0.51
HBCM area				
	TSBp	13.58 ± 0.26	96.62 ± 1.00	5.74 ± 0.09
	L11	7.04 ± 0.67	88.34 ± 1.27	10.63 ± 0.46
	CS	13.72 ± 0.52	95.02 ± 2.21	5.28 ± 0.15
	CH	11.52 ± 0.56	93.94 ± 1.63	7.27 ± 0.28
	SFR	10.69 ± 0.20	91.94 ± 1.05	7.70 ± 0.17
	CT	14.01 ± 0.30	96.58 ± 1.22	4.30 ± 1.88
	TV	10.73 ± 0.68	92.88 ± 1.83	7.84 ± 0.34
	SFH	11.07 ± 0.22	92.98 ± 0.76	7.53 ± 0.12
	L6	11.09 ± 0.55	92.58 ± 1.87	7.43 ± 0.23
	CR	13.02 ± 0.45	95.18 ± 1.34	6.02 ± 0.20
	CA	12.79 ± 0.46	94.66 ± 1.10	6.14 ± 0.27

Table A.3. HBCM-auto performances on the whole set of domains.

Domain	Reward	Success	Turns
TSBp	13.25 ± 0.41	95.26 ± 2.19	5.80 ± 0.09
L11	7.42 ± 0.76	87.06 ± 1.57	10.00 ± 0.55
CS	13.58 ± 0.24	95.30 ± 1.81	5.49 ± 0.22
CH	11.42 ± 1.05	93.25 ± 3.90	7.24 ± 0.29
SFR	10.57 ± 0.26	91.35 ± 1.32	7.70 ± 0.12
CT	14.19 ± 0.52	97.15 ± 3.47	5.24 ± 0.26
TV	7.60 ± 0.33	76.22 ± 2.01	7.65 ± 0.13
SFH	7.46 ± 0.28	77.78 ± 0.57	8.10 ± 0.23
L6	8.49 ± 0.46	80.02 ± 1.54	7.51 ± 0.28
CR	10.48 ± 0.31	82.98 ± 1.48	6.11 ± 0.06
CA	12.50 ± 0.40	94.65 ± 1.99	6.42 ± 0.17