Infinite-Mixture Policies in Reinforcement Learning

Unendliche Mischverteilungen als Modelle in Reinforcement Learning
Bachelor-Thesis von Ana Carolina Borg Ferreira aus São Paulo
Tag der Einreichung:

1. Gutachten: M.Sc. Hany Abdulsamad
2. Gutachten: Prof. Dr. Jan Peters
3. Gutachten: Prof. Dr. -Ing. Ulrich Konigorski

TECHNISCHE
UNIVERSITÄT
DARMSTADT

IAS

Infinite-Mixture Policies in Reinforcement Learning
Unendliche Mischverteilungen als Modelle in Reinforcement Learning

Vorgelegte Bachelor-Thesis von Ana Carolina Borg Ferreira aus São Paulo

1. Gutachten: M.Sc. Hany Abdulsamad
2. Gutachten: Prof. Dr. Jan Peters
3. Gutachten: Prof. Dr. -Ing. Ulrich Konigorski

Tag der Einreichung:

Erklärung zur Bachelor-Thesis

Hiermit versichere ich, die vorliegende Bachelor-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 27. September 2018

(Ana Carolina Borg Ferreira)

Abstract

Recent advancements in the field of Reinforcement Learning can be viewed in the context of constructing ever more efficient Stochastic Search algorithms [1][2]. Such approaches formulate the learning problem as a Stochastic Program [3] and perform sample-based iterative optimization from an initial point and converge to a local optimum. However, while recognizing the great success of such techniques, most recent approaches focus on finding a single solution to solve the task at hand, which represents only a point of the overall solution space. In this work, we aim to build on previous concepts [4][5], that aimed at finding diverse solutions by optimizing over multiple policies at the same time. Our focus will be on introducing powerful Bayesian non-parametric representations, such as Infinite-Mixture Models known as Dirichlet Processes [6] into the optimization problem. Such representations, on the one hand, have the great advantage of allowing for direct exploration in the solution space, while on other hand introducing technical difficulties in the update step that require applying Bayesian Inference approaches [7][8]. The thesis aims to address these issues and compare to state-of-the-art on toy problems.

Acknowledgments

I would like to thank my supervisor Hany Abdulsamad for his guidance and support.
I would also like to thank Prof. Jan Peters, who introduced the department of Intelligent Autonomous Systems in a very pleasant and open way.

Contents

Figures and Tables

List of Figures

List of Tables

1 Introduction

The usage of robots in real world scenarios requires the robotic agents to autonomously learn new tasks and adapt to new and unpredictable environments. The usual way for a robot to learn new skills is through reinforcement learning (RL) methods, in which the learning is done through trial and error by evaluating the reward or cost of an action at a given configuration of both robot and environment. Moreover, the robot must be able to generalize its set of skills to be able to solve a wide range of tasks in different environments.

Implementing reinforcement learning to solve robot tasks imposes many challenges that must be carefully addressed during design phase. Usually, real robots operate in a very high dimensional and continuous state-action space that are hard to manage in most reinforcement learning methods. Additionally, this state-action space should not be arbitrarily explored as some transitions might be prohibitive due to robot or environment damaging. Moreover, the evaluation of the algorithms on real robots is very resource intensive, requiring data and sample efficiency.

However, robot set-ups also presents some advantageous characteristics over other classical reinforcement learning problems. First, robot tasks can be often decomposed in a sequence of elemental movements, often known as movement primitives [9], that can each represent a pre-structures sub-policy for the given task. Movement primitives rely on the temporal correlations that motor tasks exhibits, modelled by non-linear dynamical systems for each robot movement. Second, an initial sub-optimal estimation of the policy parameters could be given by imitation learning from an expert's demonstration, being further optimized by local reinforcement learning methods.

Lastly, many motor tasks can be solved in a variety of ways, often incompatible to each other. Being able to represent and identify all of the solutions as separate sub-policies increases the flexibility and robustness of the robotic agent and its overall policy, as some of these solutions might not be physically possible to execute at some environment conditions or even due to damages to the robot. This work focus on finding diverse solutions by optimizing multiple sub-policies at the same time with non-parametric representations, in order to be able to learn multiple solutions without previous knowledge on the number of components involved. This is done by introducing infinite mixture models known as Dirichlet Processes to represent our policy and the Relative Entropy Policy Search (REPS) algorithm [1] as our learning algorithm for the optimization of the policies.

2 Related Work

In real robot learning applications, Policy Search (PS) methods are often preferable over another RL methods that requires the estimation of a value function, as this method requires the robot agent to fill a state-action space with samples, which is very costly and many times infeasible in most robot set-ups. Thus, PS methods that locally optimize initial sub-optimal solutions are widely used in robot learning tasks.

The base of our local learning algorithm will be the Relative Entropy Policy Search (REPS) algorithm (Peter et al., 2010). REPS is an Information Theoretic Policy Search that bounds the loss of information between each policy update, allowing thus the control of the exploitation-exploration trade-off without wide exploration in action space. This bound leads to a much more stable learning process while keeping the robotic environment safe from damages.

In order to be able to represent and optimize on multiple sub-policies, a hierarchical formulation such as in the Hierarchical REPS (HiREPS) algorithm (Daniel et al., 2016) is needed, in which sub-policies that represents different solutions to a task are selected according to a gating policy. This hierarchical structure based on a mixture model of sub-policies will be the base of the model developed in this work.

Being able to represent multiple policies does not however necessarily leads to a proper learning of all of these solutions. Therefore, HiREPS also adds additional constraints in order to properly separate the policies in the solution space and ensure that the policy search algorithm does not average over multiple solutions.

Although it is possible to keep all of the sub-policies in the solution space properly separated in the HiREPS algorithm, its learning process is not hierarchical, which may lead HiREPS to stop optimizing over all of the sub-policies except one or a couple. An alternative to deal with this issue is proposed in the Layered Direct Policy Search (LaDiPS) algorithm (End et. al), a episodic hierarchical policy search algorithm that combines both the hierarchical policy structure as well as a hierarchical learning process, optimizing both the sub-policy layer consisting of its mixture components and the gating policy layer. This model improves the control of properties such as the diversity and the individual learning rates of the gating and sub-policies, enabling the maintenance of several sub-policies across the learning process.

This thesis will build on hierarchical concepts from HiREPS algorithm, introducing the non-parametric infinite mixture model known as Dirichlet Process in order to extend and generalize the current model in HiREPS. Before introducing the new algorithm, the next chapter will cover the basic concepts on Relative Entropy Policy Search, Hierarchical Policy Search and Dirichlet Processes.

3 Foundations

3.1 Relative Entropy Policy Search Method

The main idea of information theoretic policy search is to bound the distance between the old policy $q(\boldsymbol{\theta})$ and the newly estimated policy $\pi(\boldsymbol{\theta})$ at each update step in order to avoid the new trajectory distribution to concentrate around a local optima in the reward space. Limiting the information loss of the updates in a robotic environment is also important to avoid wild exploration in the action space, which might be infeasible and dangerous to some robotic set-ups. The information theoretic policy search algorithm used in this work is the Relative Entropy Policy Search (REPS) [1], that formulates the policy search problem as a constrained optimization problem that results in a closed form solution for estimating the new policy parameters.

3.1.1 The Optimization Problem

In a hierarchical setting for policy search algorithm, the optimization problem for learning upper-level policies can be formulated as maximizing the average reward $J_{\boldsymbol{\omega}}$ as follows:

$$J_{\boldsymbol{\omega}} = \int_{\boldsymbol{\theta}} \pi_{\boldsymbol{\omega}}(\boldsymbol{\theta}) \int_{\tau} p(\tau \mid \boldsymbol{\theta}) R(\tau) d\tau d\boldsymbol{\theta} = \int_{\boldsymbol{\theta}} \pi_{\boldsymbol{\omega}}(\boldsymbol{\theta}) R(\boldsymbol{\theta}) d\boldsymbol{\theta}$$

where instead of finding the lower-level policy parameters $\boldsymbol{\theta}$, we want to find the distribution over $\boldsymbol{\theta}$ defined by the parameter vector $\boldsymbol{\omega}$, allowing for direct exploration in the parameter space. Moreover, we want to bound the information loss between the newly estimated policy and the old one by limiting the Kullback–Leibler (KL) divergence. By doing so, the final constrained optimization problem can be formulated as follows:

$$\max_{\pi} \int \pi(\boldsymbol{\theta}) R(\boldsymbol{\theta}) d\boldsymbol{\theta},$$

$$\begin{aligned} \text{s.t.} \quad \epsilon &\geq \int \pi(\boldsymbol{\theta}) \log \frac{\pi(\boldsymbol{\theta})}{q(\boldsymbol{\theta})} d\boldsymbol{\theta}, \\ 1 &= \int \pi(\boldsymbol{\theta}) d\boldsymbol{\theta}. \end{aligned}$$

It is shown in previous work [9] how this constrained optimization problem can be solved by using Lagrangian multipliers, resulting in a closed-form solution for the new policy:

$$\pi(\boldsymbol{\theta}) \propto q(\boldsymbol{\theta}) \exp\left(\frac{R(\boldsymbol{\theta})}{\eta}\right)$$

where η is the Lagrange multiplier related to the KL-bound constrain.

3.1.2 The Dual Function

The Lagrange multiplier η can be obtained by minimizing the dual function $g(\eta)$ of the constrained optimization problem:

$$g(\eta) = \eta\epsilon + \eta \log \int q(\boldsymbol{\theta}) \exp\left(\frac{R(\boldsymbol{\theta})}{\eta}\right) d\boldsymbol{\theta}$$

where the integral in the dual function can be approximated by samples.

3.1.3 New Policy Estimation

The new policy $\pi(\boldsymbol{\theta})$ is estimated by evaluating the reward $R(\boldsymbol{\theta})$ at each sample. We can then fit a parametric distribution $\pi_{\boldsymbol{\omega}}(\boldsymbol{\theta})$ by a weighted maximum-likelihood estimate on the samples with the following weights:

$$d_i = \exp\left(\frac{R(\boldsymbol{\theta})}{\eta}\right)$$

Using Gaussians to model our policies, we can update the parameters by a weighted mean and covariance given by the following equations:

$$\begin{aligned}\boldsymbol{\mu} &= \frac{\sum_{i=1}^{N} d_i \mathbf{x}_i}{\sum_{i=1}^{N} d_i},\\ \boldsymbol{\Sigma} &= \frac{\sum_{i=1}^{N} d_i (\mathbf{x}_i - \boldsymbol{\mu})(\mathbf{x}_i - \boldsymbol{\mu})^T}{Z}.\end{aligned}$$

where

$$Z = \frac{\left(\sum_{i=1}^{N} d_i\right)^2 - \sum_{i=1}^{N} (d_i)^2}{\sum_{i=1}^{N} d_i}$$

is used to obtain an unbiased estimate of the covariance.

3.2 Hierarchical Policy Search for Robot Reinforcement Learning

In this section, the hierarchical policy formulation where the robotic agent learns both a gating policy and sub-policies, will be further explained. This model is relevant for some complex tasks that often requires multiple sub-policies to represent different solutions to a single task, in order to increase the flexibility and robustness of the final policy. In general, the goal is to find an optimal policy that maximizes the average reward:

$$J(\pi) = \int \pi(\boldsymbol{a}) R_{\boldsymbol{a}} d\boldsymbol{a}$$

In HiREPS [4], the inference of a hierarchical policy is formulated as a latent variable optimization problem, where each latent variable represents the index of the sub-policy that generated a given action. Expectation-maximization methods can be then used to infer the new policy. In the Expectation step, the responsibilities $p(o \mid \boldsymbol{a})$of the latent variables are calculated, i.e., the probabilities that the observed data were generated by each sub-policy. Those responsibilities are then used to update the sub-policies using the previously shown REPS algorithm in the Maximization step. The hierarchical policy can be then represented as:

$$\pi(\boldsymbol{a}) = \sum_{o} \pi(o)\, \pi(\boldsymbol{a} \mid o)$$

Next, we will introduce information-theoretic constraints to incorporate the REPS algorithm in the hierarchical formulation.

3.2.1 Information Theoretic Constraints

Besides maximizing the average reward of the new policy, it is interesting to limit the loss of information between the old and new policy, which can be done by limiting the Kullback-Leibler (KL) divergence between the observed data $q(a)$ and the next policy $\pi(\boldsymbol{a})$, i.e.,

$$\epsilon \geq D_{KL}(\pi(\boldsymbol{a}) \| q(\boldsymbol{a})) = \int \pi(\boldsymbol{a}) \log \frac{\pi(\boldsymbol{a})}{q(\boldsymbol{a})} d\boldsymbol{a}$$

The relative entropy bound ϵ is directly responsible for controlling the exploration-exploitation rate. The policy update should converge quickly to a local optimum while not converging too fast enough to miss parts of the solutions. Selecting a suitable relative entropy bound ϵ is therefore crucial to the overall performance of the algorithm.

3.2.2 Learning Multiple Solutions

Being able to represent multiple solutions does not guarantee that the algorithm will actually find different solutions. Therefore, we must enforce the separation of the sub-policies in the solution space by introducing a limit to the expected change in the entropy H of the responsibilities of the sub-policies:

$$\kappa \geq \frac{E_{\boldsymbol{a}}\left[H\left(p\left(o \mid \boldsymbol{a}\right)\right)\right]}{E_{q(\boldsymbol{a})}\left[H\left(q\left(o \mid \boldsymbol{a}\right)\right)\right]} = \frac{\int \pi\left(\boldsymbol{a}\right) \sum_o p\left(o \mid \boldsymbol{a}\right) \log p\left(o \mid \boldsymbol{a}\right) d\boldsymbol{a}}{\int q\left(\boldsymbol{a}\right) \sum_o q\left(o \mid \boldsymbol{a}\right) \log q\left(o \mid \boldsymbol{a}\right) d\boldsymbol{a}}$$

where we can write $\tilde{\kappa} \geq E_{q(\boldsymbol{a})}\left[H\left(q\left(o \mid \boldsymbol{a}\right)\right)\right] \kappa$ to simplify the notation, resulting in the following constraint:

$$\tilde{\kappa} \geq E_{\boldsymbol{a}}\left[H\left(p\left(o \mid \boldsymbol{a}\right)\right)\right] = -\int \pi\left(\boldsymbol{a}\right) \sum_o p\left(o \mid \boldsymbol{a}\right) \log p\left(o \mid \boldsymbol{a}\right) d\boldsymbol{a}$$

When several sub-policies overlap in the solution space, there is a high uncertainty in deciding which sub-policy has generated each observation. Thus, limiting this entropy ensures that different sub-policies represents different and separate solutions.

3.2.3 Optimization Problem

To derive the resulting optimization problem, we combine all the equations and constraints derived in this section to build the following constrained optimization problem:

$$\max_\pi J(\pi) = \max_\pi \sum_o \int \pi\left(o\right) \pi\left(\boldsymbol{a} \mid o\right) R_{\boldsymbol{a}} d\boldsymbol{a}$$

$$\begin{array}{rrcl} \text{s.t.} & \epsilon & \geq & D_{KL}\left(\pi\left(o\right)\pi\left(\boldsymbol{a} \mid o\right) \,\|\, q(\boldsymbol{a}) p\left(o \mid \boldsymbol{a}\right)\right), \\ & \tilde{\kappa} & \geq & E_{\boldsymbol{a}}\left[H\left(p\left(o \mid \boldsymbol{a}\right)\right)\right], \\ & 1 & = & \sum_o \int \pi\left(o\right) \pi\left(\boldsymbol{a} \mid o\right) d\boldsymbol{a}. \end{array}$$

The focus of the remaining of this thesis is to replace this hierarchical formulation with a representation that does not require the definition of the number of options inside the mixture model that represents the upper-level policy. Therefore, in the next section, we will introduce a non-parametric representation that will allow us to replace our finite mixture model (FMM) by an infinite mixture model, know as Dirichlet processes.

3.3 Dirichlet Process

Dirichlet processes are a family of stochastic processes commonly used in Bayesian non-parametric models, whose draws are Dirichlet distributed probability distributions. In other words, it is a distribution over distributions typically used as a conjugate prior for discrete distributions that are infinite and non-parametric, i.e. can not be described by a finite number of parameters. With this prior, we can describe and infinite model with an unknown number of components, letting those grow freely with the observed data. $G \sim DP(\alpha_0, G_0)$ denotes a Dirichlet Process (DP) and can be specified by two parameters:

- Base distribution G_0, which is basically the mean of the DP.
- Concentration parameter α_0, which can be interpreted as an inverse-variance of the DP. The larger the α_0, the smaller the variance, and the DP's mass will concentrate more around its mean. This parameter is also know as strength parameter, as it can also be seen as the strength of the DP prior over distributions measured in units of mass (or sample size) of observations.

With this definition, it is possible to extend the finite mixture model used in previously developed Hierarchical Policy Search methods to a infinite mixture model [10] and therefore overcome the difficulties and limitations involved in defining the number of components prior to learning.

3.3.1 Dirichlet Process Construction and Representations

The intuition of a DP and its main properties can be better understood by describing the process through three different metaphors that will be briefly summarized in the rest of this section.

Pólya Urn Model

The Pólya urn model describes a method to generate independent and identically distributed (i.i.d.) random variables θ_n distributed according to G.
In an urn model, we assume G_0 to be a distribution of colors; balls correspond to data points whereas colors represent each probability distribution θ_n, which we can interpret as a cluster. The process goes as follows:

1. Start with an empty urn.
2. With probability $\propto \alpha_0$, draw $\theta_n \sim G_0$ and add a ball of that color into the urn.
3. With probability $\propto n-1$ (i.e., the current number of balls in the urn), pick a ball at random in the urn, record its color θ_n and return the ball into the urn while adding a second ball of the same observed color into the urn.

This method can be summarized as a way to predict a cluster for a new sample based on the existing sample's assignments through the following conditional distribution:

$$\theta_n \mid \theta_{1:n-1} \sim \frac{\alpha_0 G_0}{\alpha_0 + n - 1} + \frac{\sum_{i=1}^{n-1} \delta_{\theta_i}}{\alpha_0 + n - 1} \tag{3.1}$$

where δ_{θ_i} is a point mass located at θ_i.
This predictive distribution reveals one of the important properties of DPs. The point masses located at $\theta_{1:n}$ suggests that draws from G will take with some positive probability the same value, implying that G itself is composed of a sum of point masses, i.e. it is a discrete distribution.

Chinese Restaurant Process

The above described method gives a first intuition on the clustering property of the DP that can be further explored with another metaphor called the Chinese Restaurant Process (CRP), that defines a distribution of the set of samples $\{1,\ldots,n\}$ into $K < n$ clusters. Each cluster k contains a partition of the set of samples given by $\theta_{1:n}$, where all the θ_i's are equal to θ_k.
The metaphor can be described by the following steps:

1. Assume a Chinese restaurant with an infinite amount of empty tables, representing clusters, each of which can seat an infinite number of customers, representing data samples.
2. The first customer sits at the first table.
3. Customer n sits at an occupied table k with probability $n_k/\alpha_0 + n - 1$, where n_k is the number of customers currently sitting at table k, or sits at a new table $K+1$ with probability $\alpha_0/\alpha_0 + n - 1$.

This process can also be used to generate parameters for each clusters from a base distribution G_0 based on the current assignments as follows:

$$\theta_n \mid \theta_{1:n-1} = \begin{cases} \theta_k & \text{with probability } \frac{n_k}{\alpha_0 + n - 1} \\ \theta_{K+1} & \text{with probability } \frac{\alpha_0}{\alpha_0 + n - 1} \end{cases} \tag{3.2}$$

where $\theta_{K+1} \sim G_0$ represents a new draw from G_0.
This process illustrates a particularly import property of the DP, that is the rich-gets-richer phenomenon. The larger n_k is, the higher the probability of a new data sample to join it, and thus making larger clusters grow faster. This results in only a limited number of clusters being occupied although there are an infinite amount of them available, ensuring the feasibility of sampling from a DP mixture, as will be shown in next sections.
Another interesting aspect to note is how α_0 affects the dispersion of the samples by directly controlling the number of clusters, as the probability of a new cluster being created is proportional to α_0, with larger α_0 leading to a higher number of clusters a priori.

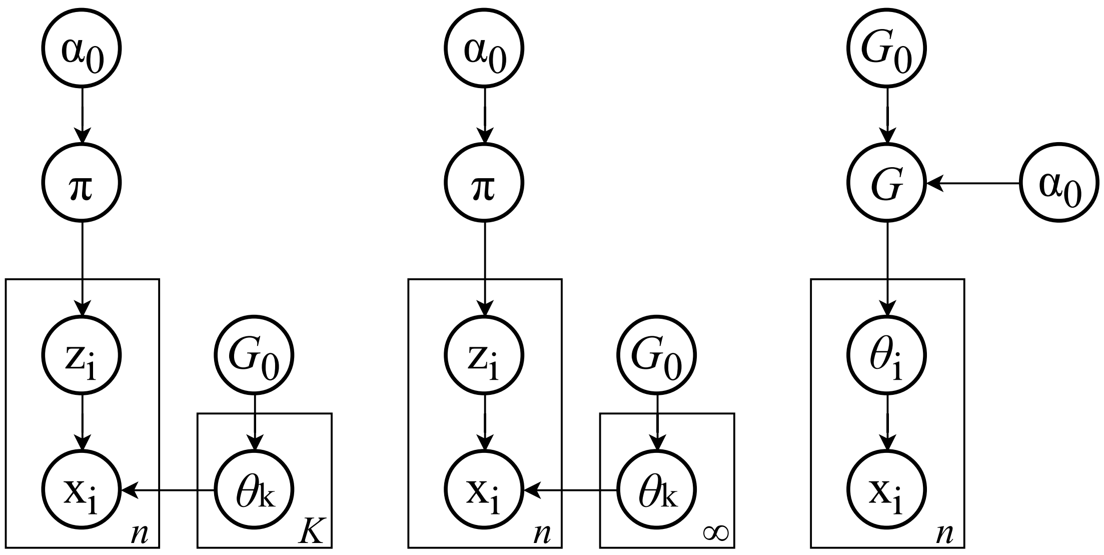

Figure 3.1: Finite Mixture Model (left), Dirichlet Process Mixture Model in stick-breaking representation (center) and Dirichlet Process Mixture Model in Pólya urn representation (right)

Stick-breaking Process

The above representations showed both the discrete and clustering properties of the DP by taking draws from G. The stick-breaking process is a constructive method that explicitly shows that G is composed of a weighted sum of point masses:

$$G = \sum_{k=1}^{\infty} \pi_k \delta_{\theta_k} \tag{3.3}$$

where $\theta_k \sim G_0$. The mixture weights $\pi = (\pi_k)_{k=1}^{\infty}$ can be constructed as follows:

1. Start with a stick of length 1 and break it at $\beta_1 \sim \text{Beta}(1, \alpha_0)$, assigning the broke off stick length to π_1.
2. Recursively break the remaining stick at $\beta_k \sim \text{Beta}(1, \alpha_0)$ to obtain the remaining weights $\pi_1, \pi_2, \dots, \pi_k$.

The process can be summarized as follows:

$$\begin{aligned} \beta_k &\sim \text{Beta}(1, \alpha_0) \\ \pi_k &= \beta_k \prod_{i=1}^{k-1} (1 - \beta_i) \end{aligned} \tag{3.4}$$

The stick-breaking distribution $\pi = (\pi_k)_{k=1}^{\infty}$ satisfies $\sum_{k=1}^{\infty} \pi_k = 1$ and can be written as $\pi \sim \text{GEM}(\alpha_0)$, named after Griffiths, Engen and McCloskey.

3.3.2 Dirichlet Processes Mixture Model

A Dirichlet Process Mixture Model (DPMM) can be viewed as an extension of the finite mixture model (FMM) previously used in Hierarchical Policy Search methods when the number of components go to infinity [10]. For that reason, the FMM will be first described before considering the infinite model and its implications.
A graphical representation of a FMM is given in Figure 3.1, which can be equally described by the following probability distributions:

$$\pi \mid \alpha_0 \sim \text{Dirichlet}(\alpha_0/K, \dots, \alpha_0/K) \tag{3.5}$$
$$z_i \mid \pi \sim \text{Multinomial}(\pi) \tag{3.6}$$
$$\theta_k \sim G_0 \tag{3.7}$$
$$x_i \mid z_i, \{\theta_k\}_{k=1}^{K} \sim F\left(\theta_{z_i}\right) \tag{3.8}$$

In a FMM, the mixture weight π of each component is sampled from a symmetric Dirichlet prior with concentration parameter α_0 as in (5), which is used to parametrize the multinomial distribution from which the components will be selected as in (6). The component associated to each data point x_i is specified by a indicator variable $z_i \in \{1, \ldots, K\}$. The data point x_i is then generated by sampling from the distribution $F\left(\theta_{z_i}\right)$ parametrized by the component parameters θ_{z_i} as in (8). The component parameters θ_k are sampled from a common conjugate prior distribution G_0.
When it is not possible to determine the fixed number K of components, the FMM should be replaced by a DPMM by letting K go to infinity, as shown by the graphical representation in Figure 3.1. To generate the the DPMM, the Dirichlet prior for the mixture weights vector is replaced by the stick-breaking construction $\boldsymbol{\pi} \sim \mathrm{GEM}(\alpha_0)$, and the new conditional distributions for the DPMM are given as follows:

$$\begin{aligned} \boldsymbol{\pi} \mid \alpha_0 &\sim \mathrm{GEM}(\alpha_0) \\ z_i \mid \boldsymbol{\pi} &\sim \mathrm{Multinomial}(\boldsymbol{\pi}) \\ \theta_k &\sim G_0 \\ x_i \mid z_i, \{\theta_k\}_{k=1}^{\infty} &\sim F\left(\theta_{z_i}\right) \end{aligned} \tag{3.9}$$

By omitting the indicator variables, we can get the DPMM in the Pólya urn representation by letting $\theta_i = \theta_{z_i}$, expliciting the generative process of the component parameters. The graphical representation is in Figure 3.1 and the equivalent conditional probabilities are the following:

$$\begin{aligned} G \mid G_0, \alpha_0 &\sim G_0 \\ \theta_i &\sim G \\ x_i \mid \theta_i &\sim F(\theta_i) \end{aligned} \tag{3.10}$$

The conditional distributions of the indicator variables z_i can be derived for the FMM [10]:

$$P(z_i = k \mid \mathbf{z}_{-i}, \alpha_0) = \frac{n_{k,-i} + \alpha_0/K}{\alpha_0 + n - 1} \tag{3.11}$$

where $\mathbf{z}_{-i}$ represents the number of data points assigned to the k-th component excluding the i-th point. By letting K go to infinity, the conditional distributions reaches the following limits:

$$\begin{aligned} P(z_i = k \mid \mathbf{z}_{-i}, \alpha_0) &= \frac{n_{k,-i}}{\alpha_0 + n - 1}, \text{ for cluster k with } n_{k,-i} > 0 \\ P(z_i = K + 1 \mid \mathbf{z}_{-i}, \alpha_0) &= \frac{\alpha_0}{\alpha_0 + n - 1}, \text{ for new cluster} \end{aligned} \tag{3.12}$$

The above equations illustrate how the DPMM is closely related to the previously described Chinese Restaurant Process, as Equations (3.12) and 3.(2) are equivalent. Therefore, the rich-gets-richer property of the Dirichlet Process and the influence of the concentration parameter α_0 also apply to the infinite mixture model.

The DPMM gives us a flexible mixture model in which the number of components grows with the observed data, but the difficulty remains on fitting the component parameters by inferring its posterior distribution. Usually, Expectation Maximization (EM) algorithm is used for inference in a mixture model, but the non-parametric nature of G makes it difficult to implement. One usual way of dealing with this issue is to use a Monte-Carlo Markov chain (MCMC) sampling method called Gibbs Sampling, which will be derived for the DPMM in the next section.

3.3.3 Gibbs Sampling Method for Dirichlet Process Mixtures

Gibbs Sampling is a Markov chain Monte Carlo (MCMC) algorithm used when sampling from a multivariate joint distribution directly is difficult or the distribution itself is not know explicitly, but the conditional probability of each variable is easier to sample from. The Gibbs sampler generates a Markov chain of samples, in which samples of each variable are iteratively drawn conditionally on the current values of other variables.

Although implementing a Gibbs sampler on a DPMM based on the Pólya urn representation (10) may seem easier, as the only unknown variables are $\{\theta_i\}_{i=1}^{n}$ it is usually very inefficient. With this method, we would draw from the posterior distribution of θ_i conditioned on all the other variables $\boldsymbol{\theta}_{-i}$ and the observations. This means that, at each iteration, the component parameters must be sampled n times in order to update the parameters of a single data point. As the data points are grouped in a finite number of components, a more efficient way to implement the Gibbs sampler is to operate on all of the data points assigned to each mixture component simultaneously. This method is easily implemented when using the stick-breaking representation of DPMMs (3.9), in which indicator variables $\mathbf{z}$ are used to identify at which component each data point x_i is assigned and the component parameters are related to each indicator variable instead of each data point, as illustrated in Figure 3.1.

Before deriving the Gibbs sampling method for DPMM, it is easier to start with the FMM case. The Gibbs sampler will alternately draw samples from each of the variables while keeping all the others fixed: the indicator variables $\mathbf{z} = \{z_i\}_{i=1}^n$, the mixture weights $\boldsymbol{\pi}$ and the parameters of the mixture components $\boldsymbol{\theta} = \{\theta_k\}_{k=1}^K$, while the concentration parameter α_0 and the parameters of the base distribution G_0 are assumed to be known . Thus, the first step to implement the Gibbs sampler is to derive the conditional posterior distribution for all these variables by exploiting both Bayes rule and the inherent Markov properties of FMMs.

First, we derive the conditional posterior distribution for each indicator variable z_i:

$$\begin{aligned}
p\left(z_i = k \mid \mathbf{z}_{-i}, \mathbf{x}, \{\theta_k\}_{k=1}^K, \boldsymbol{\pi}, \alpha_0\right) &= p\left(z_i = k \mid x_i, \{\theta_k\}_{k=1}^K, \boldsymbol{\pi}\right) && (3.13)\\
&\propto p\left(z_i = k \mid \{\theta_k\}_{k=1}^K, \boldsymbol{\pi}\right) p\left(x_i \mid z_i = k, \{\theta_k\}_{k=1}^K, \boldsymbol{\pi}\right) && (3.14)\\
&= p\left(z_i = k \mid \boldsymbol{\pi}\right) p\left(x_i \mid \theta_k\right) && (3.15)\\
&= \pi_k f\left(x_i \mid \theta_k\right)
\end{aligned}$$

In this derivation, the Markov property of the FMM is explored in (13), (14) uses the Bayes rule that posterior $\propto$prior $\times$ likelihood and (15) uses the definition of indicator variables and the Markov property again.

For the mixture weights $\boldsymbol{\pi}$, we derive its conditional posterior:

$$\begin{aligned}
p\left(\boldsymbol{\pi} \mid \mathbf{z}, \mathbf{x}, \{\theta_k\}_{k=1}^K, \alpha_0\right) &= p\left(\boldsymbol{\pi} \mid \mathbf{z}, \alpha_0\right) && (3.16)\\
&= \mathrm{Dir}\left(n_i + \alpha_0/K, \ldots, n_K + \alpha_0/K\right) && (3.17)
\end{aligned}$$

where $n_k = \sum_{i=1}^n \delta\left(z_i - k\right)$. In the above derivation, (3.16) results from Markov property and (17) uses the property of the conjugate Dirichlet prior.
The conditional posterior for the component parameters need also to be derived. The mixture parameters $\{\theta_k\}_{k=1}^K$ and weights $\boldsymbol{\pi}$ are mutually independent conditioning on the latent indicator variables $\mathbf{z}$:

$$p\left(\boldsymbol{\pi}, \{\theta_k\}_{k=1}^K \mid \mathbf{z}, \mathbf{x}, \alpha_0\right) = p\left(\boldsymbol{\pi} \mid \mathbf{z}, \alpha_0\right) \prod_{k=1}^K p\left(\theta_k \mid \mathbf{x}_k\right) \qquad (3.18)$$

This result shows how the conditional posterior of θ_k depends only on the observations $\mathbf{x}_k$ assigned to its k-th component, and therefore can be derived as follows:

$$\begin{aligned}
p\left(\theta_k \mid \boldsymbol{\theta}_{-k}, \mathbf{z}, \mathbf{x}, \pi, \alpha_0\right) &= p\left(\theta_k \mid \mathbf{x}_k\right) && (3.19)\\
&\propto G_0\left(\theta_k\right) f\left(\mathbf{x}_k \mid \theta_k\right) && (3.20)
\end{aligned}$$

Here, (3.19) uses the Markov property and the results from (3.18), while (3.20) uses the Bayes rule.
In (3.17), the mixture weights $\boldsymbol{\pi}$ are directly sampled from a Dirichlet distribution, which is difficult when extending the sampler to DPMMs as K go to infinity. One alternative is to derive the conditional posterior of the indicator variable z_i integrating the mixture weight π out [10]:

$$\begin{aligned}
p\left(z_i = k \mid \mathbf{z}_{-i}, \mathbf{x}, \{\theta_k\}_{k=1}^K, \alpha_0\right) &= p\left(z_i = k \mid \mathbf{z}_{-i}, x_i, \theta_k, \alpha_0\right) && (3.21)\\
&\propto p\left(z_i = k \mid \mathbf{z}_{-i}, \theta_k, \alpha_0\right) p\left(x_i \mid z_i = k, \mathbf{z}_{-i}, \theta_k, \alpha_0\right) && (3.22)\\
&= p\left(z_i = k \mid \mathbf{z}_{-i}, \alpha_0\right) p\left(x_i \mid \theta_k\right) && (3.23)\\
&= \frac{n_{k,-i} + \alpha_0/K}{\alpha_0 + n - 1} f\left(x_i \mid \theta_k\right) && (3.24)
\end{aligned}$$

where (3.21) uses the Markov property, the definition of indicator variables and the results from (3.18), (3.22) uses the Bayes rule, (3.23) the Markov property and (3.24) the results in (3.11).
With all the conditional posteriors properly derived, it is possible to extend the Gibbs sampling method for FMM to DPMM by letting K go to infinity. In this case, the conditional prior of z_i becomes (3.12) and we can use the result in (3.24) to rewrite the conditional posterior of z_i as follows, for the case when z_i is assigned to one of the existing components:

$$p\left(z_i = k \mid \mathbf{z}_{-i}, \mathbf{x}, \{\theta_k\}_{k=1}^{K}, \alpha_0\right) = \frac{n_{k,-i}}{\alpha_0 + n - 1} f\left(x_i \mid \theta_k\right)$$

We still need to derive the conditional posterior for z_i for the case when it is assigned to a new component, denoted as $K+1$:

$$\begin{aligned} p\left(z_i = K+1 \mid \mathbf{z}_{-i}, \mathbf{x}, \alpha_0\right) &= p\left(z_i = K+1 \mid \mathbf{z}_{-i}, x_i, \alpha\right) && (3.25)\\ &\propto p\left(z_i = K+1 \mid \mathbf{z}_{-i}, \alpha_0\right) p\left(x_i \mid z_i = K+1, \mathbf{z}_{-i}, \alpha_0\right) && (3.26)\\ &= p\left(z_i = K+1 \mid \mathbf{z}_{-i}, \alpha_0\right) p\left(x_i\right) && (3.27)\\ &= \frac{\alpha_0}{\alpha_0 + n - 1} \int G_0(\theta) f\left(x_i \mid \theta\right) d\theta && (3.28) \end{aligned}$$

In the above derivation, (3.25) uses the definition of indicator variables, (3.26) uses the Bayes rule, (3.27) uses both the definition of indicator variables as well as the Markov property, and (3.28) uses the result in (3.12) and the definition of marginal distribution. If z_i is assigned to a new component $K+1$, a new parameter θ_{K+1} must be drawn from the posterior distribution based on the prior G_0 and the observation x_i, increasing K by 1.
As previously investigated, the number K of components is very sensitive to α_0 [13], making it necessary to choose a weakly informative prior and learn it from the data. Applying a Gamma prior to $\alpha_0 \sim \text{Gamma}(a, b)$ makes it easy to draw samples from its posterior via auxiliary method [13].
The whole Gibbs sampling method for DPMM can be summarized in Algorithm 1.

```
Given $\left\{z_i^{(t-1)}\right\}_{i=1}^{n}$, $\left\{\theta_k^{(t-1)}\right\}_{k=1}^{K}$ and $\alpha_0^{(t-1)}$ from the previous iterations , sample $\left\{z_i^{(t)}\right\}_{i=1}^{n}$, $\left\{\theta_k^{(t)}\right\}_{k=1}^{K}$ as follows:
  1. Set $z = z^{(t-1)}$, $\alpha_0 = \alpha_0^{(t-1)}$
  2. For $i = 1, \dots, n$
      a) Remove data point $x_i$ from component related to $z_i$
      b) If the current component $z_i$ becomes empty after Step (2.a), the component and its parameter are removed, and $K$ decreases by 1
      c) Re-arrange the indicator variables $\mathbf{z}$ so that all are non-empty
      d) Assign a new $z_i$ for $x_i$ by sampling from the following probabilities
         $p(z_i = k, k \le K) \propto \frac{n_{k,-i}}{\alpha_0 + n - 1} f(x_i \mid \theta_k) \qquad n_{k,-i} = \sum_{j \ne i} \delta(z_j - k)$
         $p(z_i = K+1) \propto \frac{\alpha_0}{\alpha_0 + n - 1} \int G_0(\theta) f(x_i \mid \theta) d\theta$
      e) If $z_i = K+1$, a new parameter $\theta_{K+1}$ must be drawn from the posterior distribution based on the prior $G_0$ and the observation $x_i$, and increase $K$ by 1.
  3. For $k = 1, \dots, K$
      a) Sample parameters $\theta_k$ for each component from the following distribution:
         $\theta_k^{(t)} \propto G_0(\theta_k) f\left(\mathbf{x}_k^{(t)} \mid \theta_k^{(t-1)}\right)$
  4. Set $z^{(t)} = z$
  5. Sample $\alpha_0^{(t)} \sim p(\alpha_0 \mid K, n, a, b)$ via auxiliary variable method [13].
```

Algorithm 1: Gibbs sampling for DPMM

4 Contribution

In this chapter, the finite mixture model used in hierarchical formulations such as in HiREPS will be extended to the infinite mixture case with the use of Dirichlet Process to model the mixture components of a policy and REPS to perform the local optimization of the sub-policies.

4.1 Relative Entropy Policy Search in Infinite Mixture Model

In this chapter, the finite mixture model used in hierarchical formulations such as in HiREPS will be extended to the infinite mixture case with the use of Dirichlet Process to model the mixture components of a policy and REPS [1] to perform the local optimization of the sub-policies.

In HiREPS [4], as shown in previous chapters, a finite mixture model was used, in which the number of options representing the sub-policies must be somehow set prior to the learning, and the update of the policies parameters is made through Expectation-maximization with the weights provided by the REPS algorithm. With the proposed method in this thesis, the sub-policies will be represented by a Dirichlet Process, allowing the number of components in the mixture to grow freely with the data without any prior knowledge of the underlying structure of the overall policy. The upper-level policy is controlled by the concentration parameter α_0 of the Dirichlet Process, that is directly related to the responsibilities of each sub-policy. Finally, the sub-policies are updated via Gibbs Sampling [10] with the weights provided by the previously described REPS algorithm.

In order to properly implement the Gibbs sampler to the update of the sub-policies, we must first define the prior G_0 used to each mixture component and derive its posterior distribution based on the observed data $\mathbf{x}$, as well as the marginal likelihood of x_i. To be able to model the sub-policies in the most flexible way, the Normal-inverse-Wishart distribution was chosen as our base distribution G_0, allowing the learning of both mean and covariance matrix of each sub-policy. In the next section, the equations needed to the implementation of the Gibbs sampler will be presented.

4.1.1 Normal-inverse-Wishart Distribution as Prior

A Normal-inverse-Wishart distribution is the conjugate prior of the multivariate normal distribution, with unknown mean and covariance matrix. It is defined by four parameters NIW$(\mu_0, \kappa_0, \Lambda_0, \nu_0)$ and it is possible to generate a random variable with two basic steps [12] :

1. Sample Σ from an inverse Wishart distribution with parameters Λ_0 and ν_0
2. Sample μ from a multivariate normal distribution with mean μ_0 and covariance matrix $\frac{1}{\kappa_0}\Sigma$

In step (2.e) of the previously shown Gibbs sampling algorithm, new component parameters θ_{K+1} must be drawn from the posterior distribution based on the prior G_0 and the observation x_i. It can be shown [12] that the posterior distribution of the parameter of the Normal-inverse-Wishart distribution is given as follows:

$$\begin{aligned}
p(\mu, \Sigma \mid \mathbf{x}, \mu_0, \kappa_0, \Lambda_0, \nu_0) &= \mathrm{NIW}(\mu, \Sigma \mid \mu_n, \kappa_n, \Lambda_n, \nu_n) \\
\mu_n &= \frac{\kappa_0}{\kappa_0 + n}\mu_0 + \frac{n}{\kappa_0 + n}\bar{x} \\
\kappa_n &= \kappa_0 + n \\
\nu_n &= \nu_0 + n \\
\Lambda_n &= \Lambda_0 + S + \frac{\kappa_0 n}{\kappa_0 + n}(\bar{x} - \mu_0)(\bar{x} - \mu_0)^T
\end{aligned}$$

where $S = \sum_{i=1}^{N}(x_i - \bar{x})(x_i - \bar{x})^T$ is the scatter matrix.

In step (3.a) of the previously shown Gibbs sampling algorithm, the updated parameters of each component θ_k must be drawn from the marginal likelihood of $\mathbf{x}_k$. It can be shown [12] that this distribution is derived as follows:

$$\int G_0(\theta) f(x_i \mid \theta)\, d\theta = \frac{1}{\pi^{nd/2}} \frac{\Gamma_d(\nu_n/2)}{\Gamma_d(\nu_0/2)} \frac{|\Lambda_0|^{\nu_0/2}}{|\Lambda_n|^{\nu_n/2}} \left(\frac{\kappa_0}{\kappa_n}\right)^{d/2}$$

With those equations in hand, we can now fully implement a version of the Gibbs sampling algorithm with REPS optimization.

4.1.2 Infinite Mixture Models with Relative Entropy Policy Search

In order to optimize over the sub-policies, we need to incorporate the weights from REPS algorithm into the Gibbs sampler. Instead of directly applying the weights to the mean and covariance matrix of the sub-policies, we apply them to the parameters of the inverse scale matrix Λ right before the update step (3.a) of the Gibbs sampling algorithm.

The learning in the upper level policy is controlled by the concentration parameter α_0 of the Dirichlet Process, that affects both the quantity of components to be stay active during the learning process, as well as the responsibilities of each sub-policy. The learning of α_0 is currently being done by sampling $\alpha_0^{(t)} \sim p(\alpha_0 \mid K, n, a, b)$ via auxiliary variable method [13]. It is useful to initialize α_0 with a non-informative Gamma prior, but the sensitivity of the Gibbs sampler to the concentration parameter is too high, making the task of properly initializing and updating α_0 one of the biggest challenges in this hierarchical policy search model.

5 Experiments

In this chapter, all of the conducted experiments with the Gibbs sampler and the REPS with applied to infinite mixture models will be further detailed.

5.1 Clustering

Before trying to optimize the sub-policies with REPS, we should be able to apply the pure Gibbs sampler in order to update the mixture components parameters based on the assignments of each observed data sample. The natural first step to implement this algorithm is thus to apply it in a simple clustering problem of a Dirichlet-distributed data in order to investigate and understand its behaviour.

5.1.1 2D Clustering

The first example to be investigated is a 2D clustering problem with its data distributed according to Figure 5.1 and the means of each clusters stated in Table 5.1.

5.1.2 4D Clustering

In order to investigate the performance of the Gibbs sampler in a higher-dimensional space, another clustering example was built with 4-dimensional clusters, with its respective means presented in Table 5.1.

5.2 Dirichlet Process Mixture Model with REPS

The idea of applying REPS to the infinite mixture model is to optimize the sub-policies evaluating the contribution of each observed data sample regarding to the average reward applying weights to each of the data samples. The idea of applying REPS to the infinite mixture model is to optimize the sub-policies evaluating the contribution of each observed data sample regarding to the average reward applying weights to each of the data samples. We tested the algorithm in a 2-dimensional test bench function called Himmelblau function, illustrated in Figure 5.2. This function will act as the reward in our reinforcement learning task, and the goal is to find all of its zeros denoted in Table 5.2. The algorithm will be run for three different values of the concentration parameter α_0 and the initial data distribution while keeping the other parameters fixed.

5.3 Initial Settings

For all tests, the base Normal-inverse Wishart distribution prior is set in such a way that acts as a non-informative prior [12] to the sub-policies or clusters, i.e. NIW($\mathbf{0}, 4.5, I, d+1$), where d is the dimension of the data.

μ_0	μ_1
-5	5
-5	-5
5	5
5	-5
15	-5
15	5

μ_0	μ_1	μ_2	μ_3
-5	5	-2	2
-5	-5	-2	-2
5	5	2	2
5	-5	2	-2

Table 5.1: Clustering problem: means of clusters for 2D and 4D case

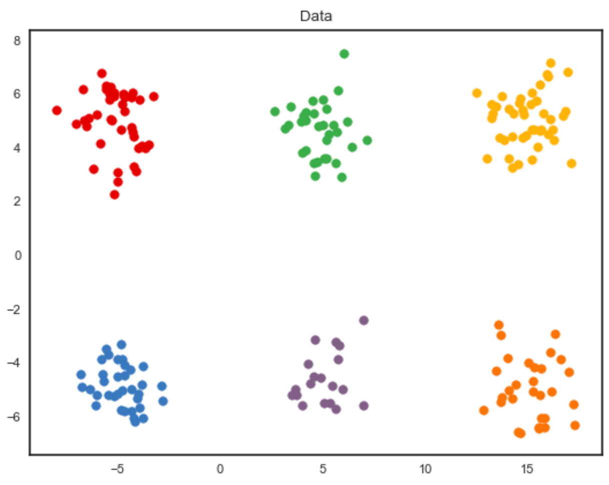

Figure 5.1: 2D Clustering problem: application of Gibbs sampler to a clustering problem in order to investigate its behavior and update the mixture components parameters

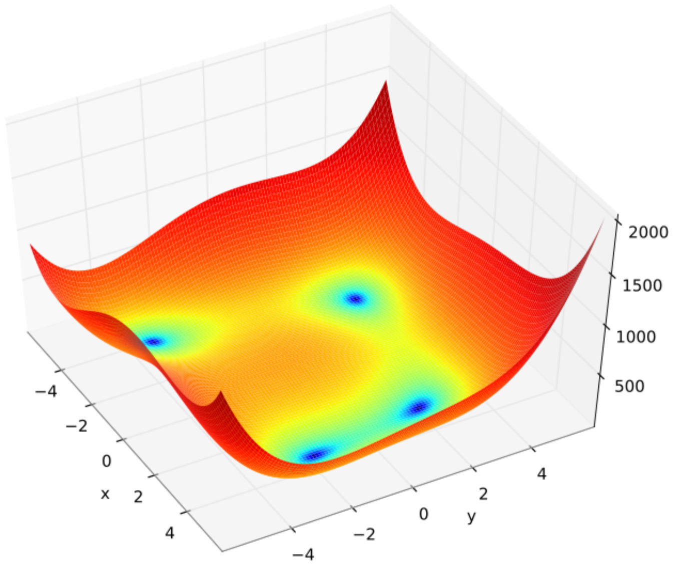

Figure 5.2: Himmelblau function: the testbench function will provide the rewards to our reinforcement learning environment, and the goal is to find all of its zeros. Image from Wikimedia Commons, the free media repository

μ_0	μ_1
3.0	2.0
-2.8	3.13
-3.78	-3.28
3.58	-1.85

Table 5.2: Zeros of the Himmelblau function

For the 2D clustering problem, the Gibbs sampler was run a total of 100 times with $n = 100$ samples.
For the 4D clustering problem, the Gibbs sampler was run a total of 100 times with $n = 200$ samples.
For the DPMM with REPS case, the optimization loop will be run 100 times, where at each iteration the Gibbs sampler will be run 150 times in order to fit the new policy parameters with $n = 500$ samples. The entropy bound ϵ is set to a low value 0.1 to avoid premature convergence. Three different values for the concentration parameter α_0 will be used: 10, 15 and 11.

6 Results

6.1 2D Clustering

The results of the 2D Clustering problem can be viewed in Figure 6.1. In the figure, only the clusters that hold more than 5% of the total data are represented. It is important to note here that the Gibbs sampler returned a higher number of clusters $K = 12$ that contained only an insignificant portion of the data, as we expected from the formulation of infinite mixture models.
It is also possible to analyse the accuracy of the clustering as well as the number of clusters across the iterations in Figure 6.2 . From this figure, it is possible to observe how fast the algorithm converges to the final accuracy and number of clusters, with less than 20 iterations needed.
In Figures 6.3 to 6.8, each of the cluster means were sampled 100 times in order to visualize the variance at each dimension.

6.2 4D Clustering

It is also possible to analyse the accuracy of the clustering as well as the number of clusters across the iterations in Figure 6.9 .
In Figures 6.10 to 6.13, each of the cluster means were sampled 100 times in order to visualize the variance at each dimension.

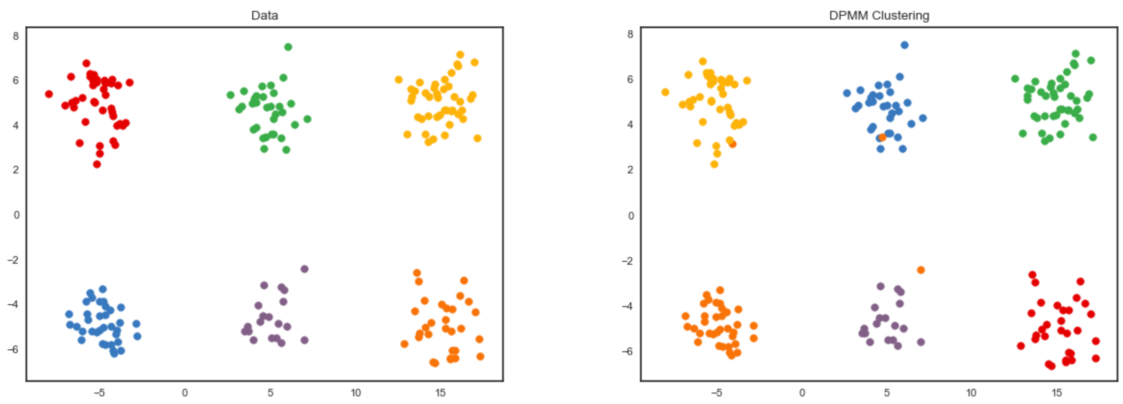

Figure 6.1: 2D Clustering problem: Comparison of initial data distribution with final clustering result

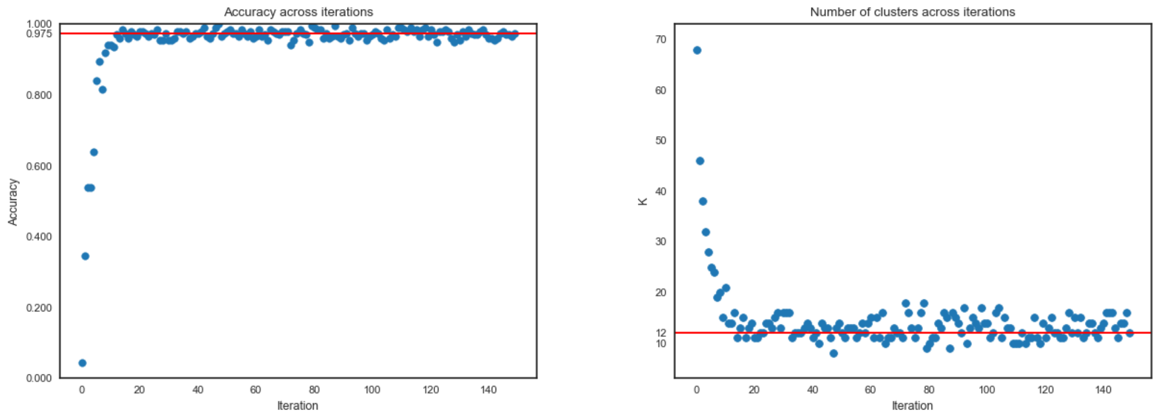

Figure 6.2: 2D Clustering problem: Accuracy and number of cluster across iterations

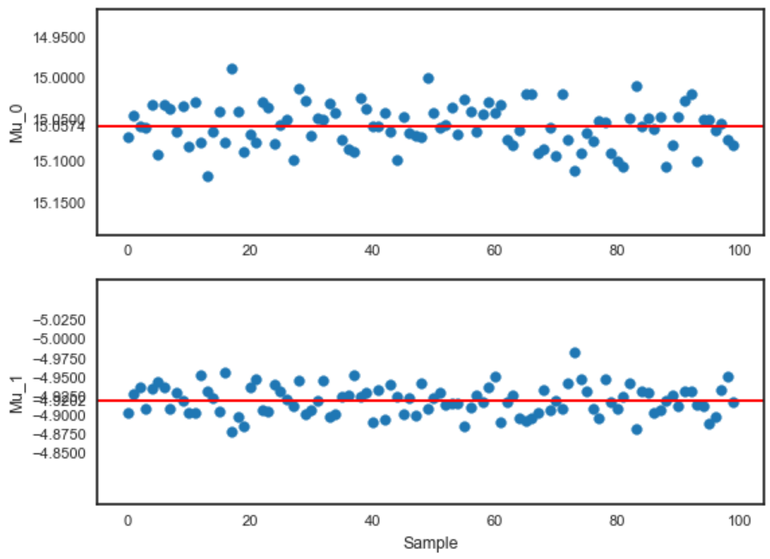

Figure 6.3: 2D Clustering problem: Mean values of cluster K_0sampled 100 times

μ_{0true}	μ_{1true}	μ_{2true}	μ_{3true}
-5	5	-2	2
-5	-5	-2	-2
5	5	2	2
5	-5	2	-2

μ_{0test}	μ_{1test}	μ_{2test}	μ_{3test}
-4.70686	5.29424	-2.30452	1.2903
-5.00184	-4.63845	-2.42868	-2.25678
5.44083	5.01058	1.66255	2.18746
4.1167	-5.15411	2.21012	-1.4298

Table 6.1: 4D Clustering problem: Comparison of true mean of the clusters with final means after Gibbs Sampling

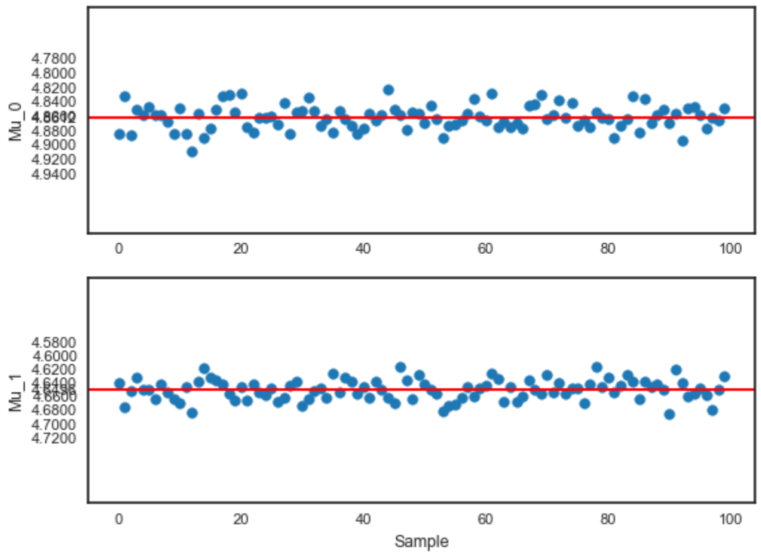

Figure 6.4: 2D Clustering problem: Mean values of cluster K_1sampled 100 times

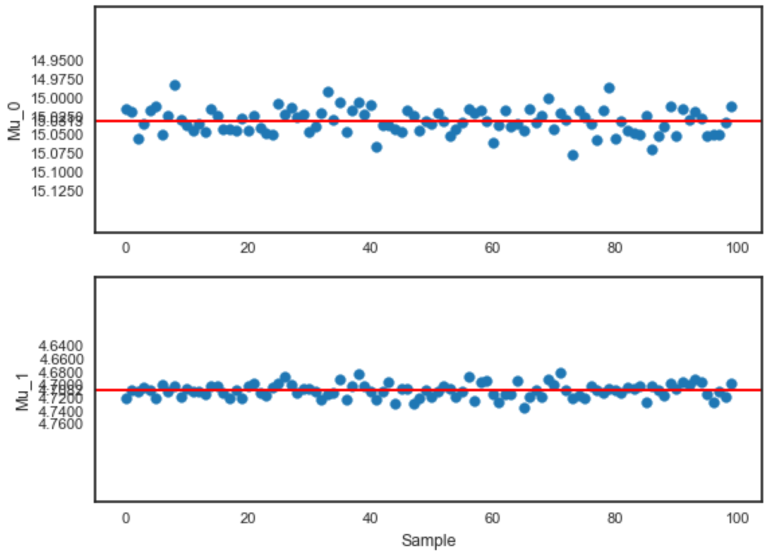

Figure 6.5: 2D Clustering problem: Mean values of cluster K_2sampled 100 times

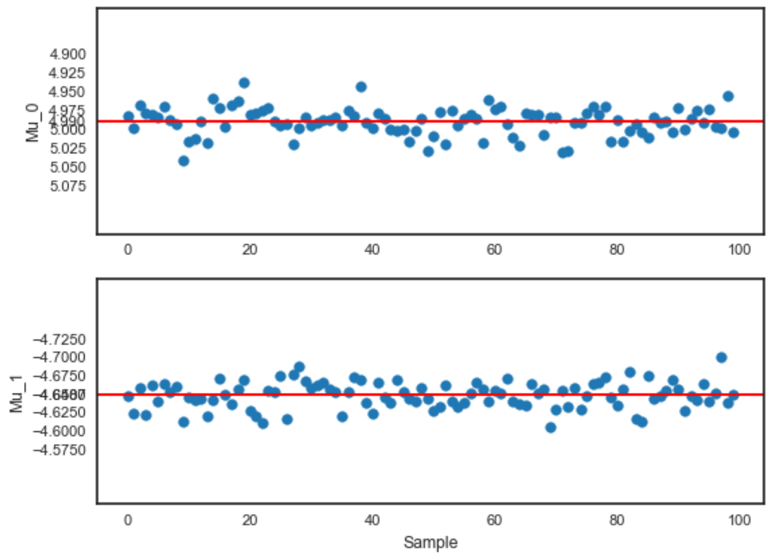

Figure 6.6: 2D Clustering problem: Mean values of cluster K$_3$sampled 100 times

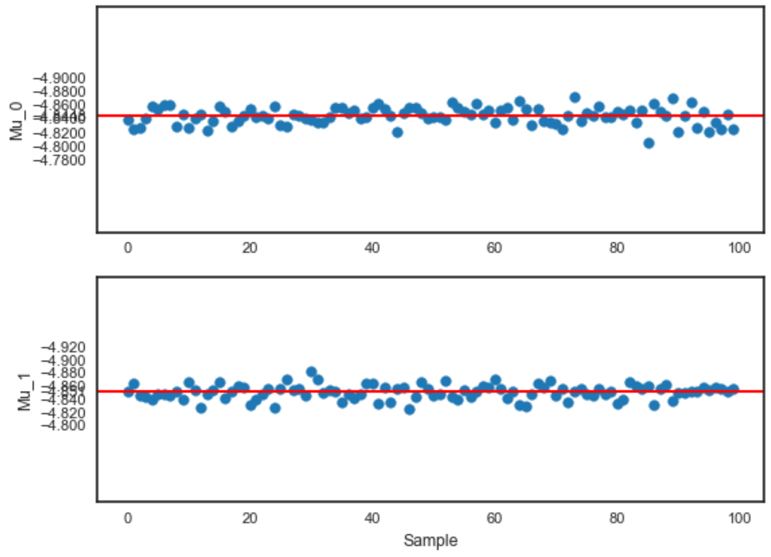

Figure 6.7: 2D Clustering problem: Mean values of cluster K$_4$sampled 100 times

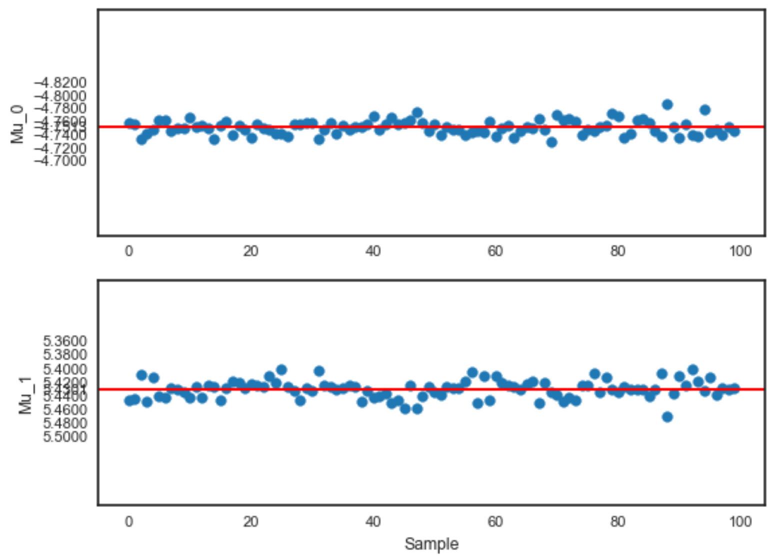

Figure 6.8: 2D Clustering problem: Mean values of cluster K_5sampled 100 times

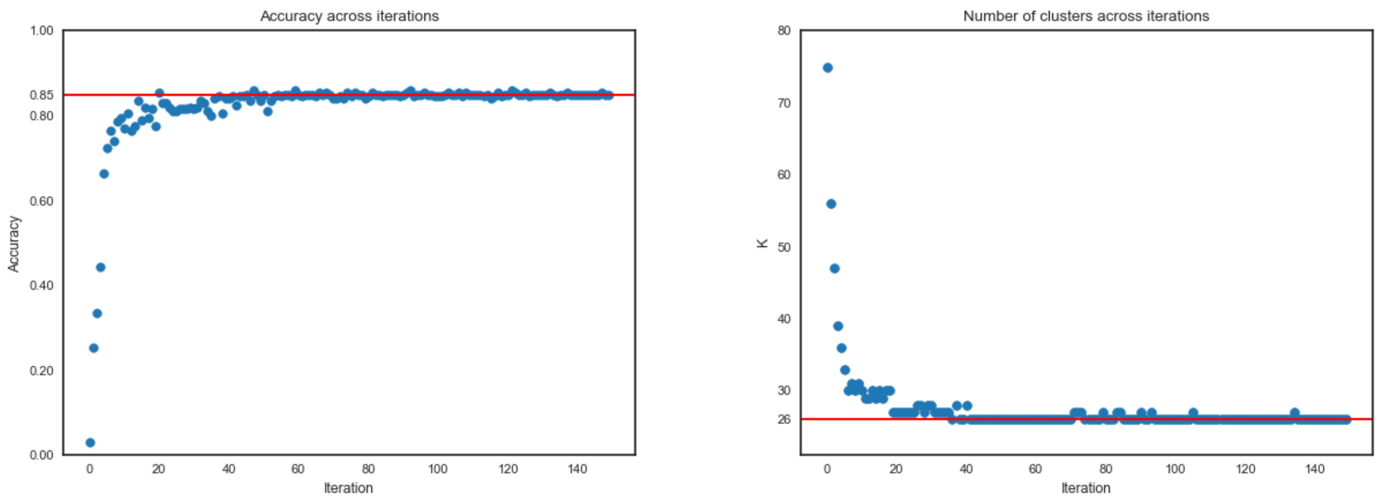

Figure 6.9: 4D Clustering problem: Accuracy and number of cluster across iterations

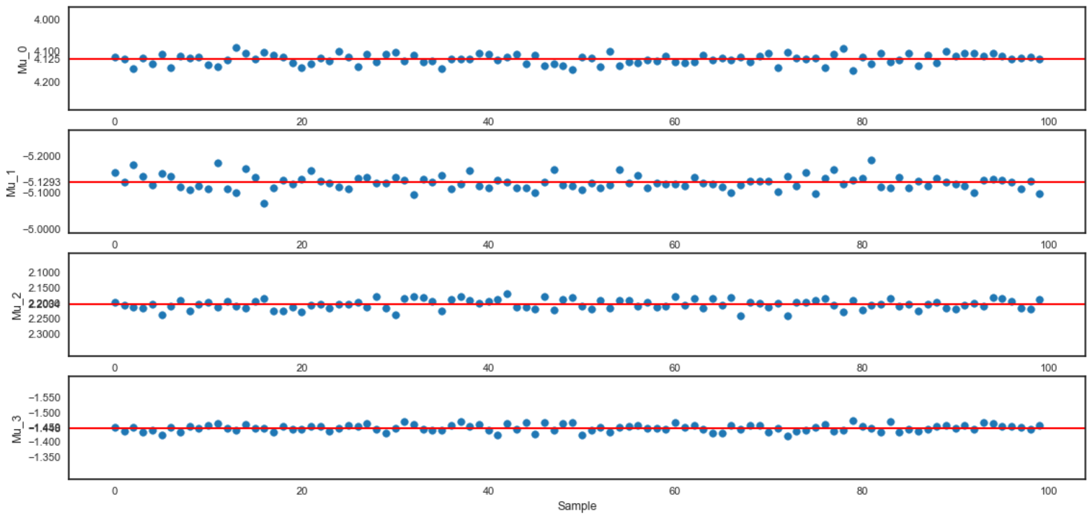

Figure 6.10: 4D Clustering problem: Mean values of cluster K_0sampled 100 times

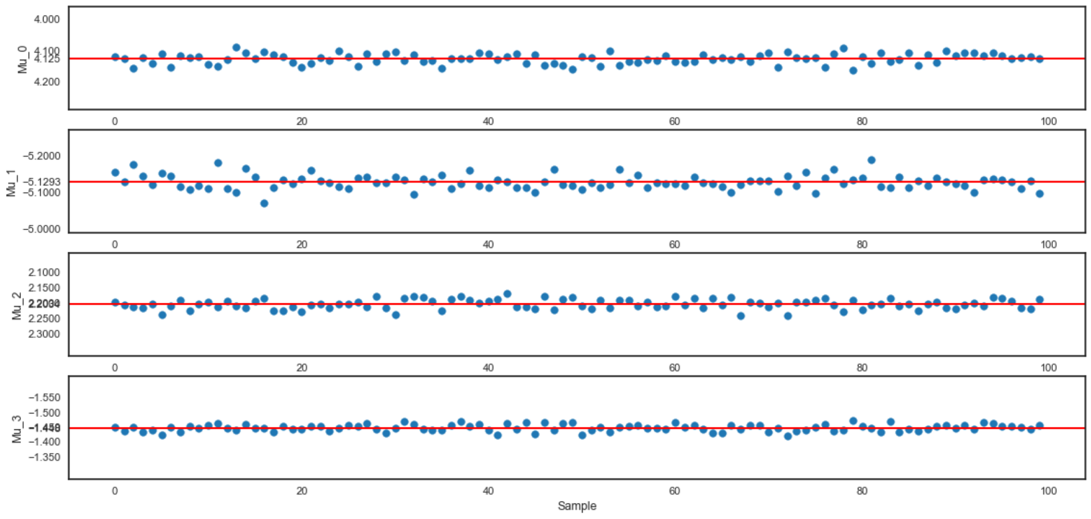

Figure 6.11: 4D Clustering problem: Mean values of cluster K_1sampled 100 times

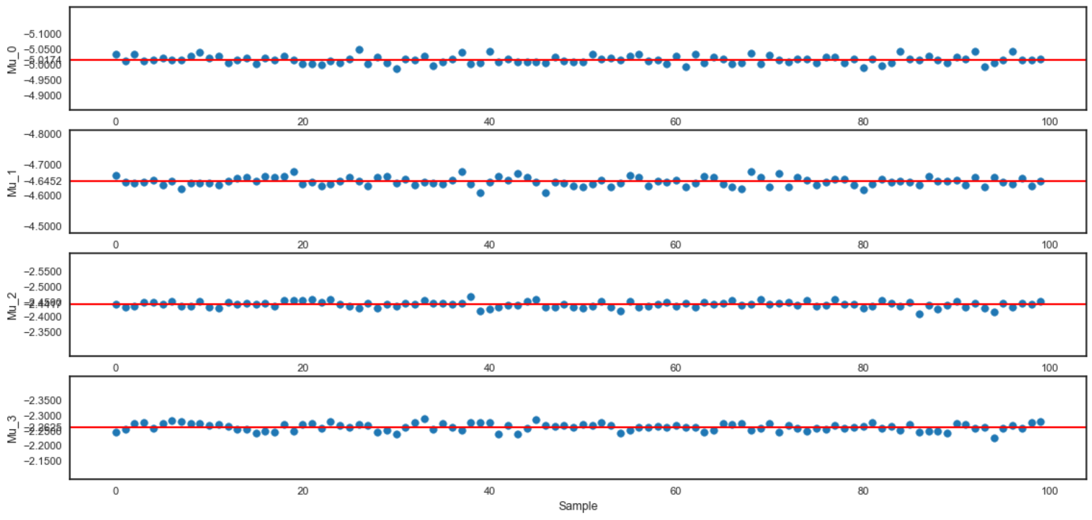

Figure 6.12: 4D Clustering problem: Mean values of cluster K_2sampled 100 times

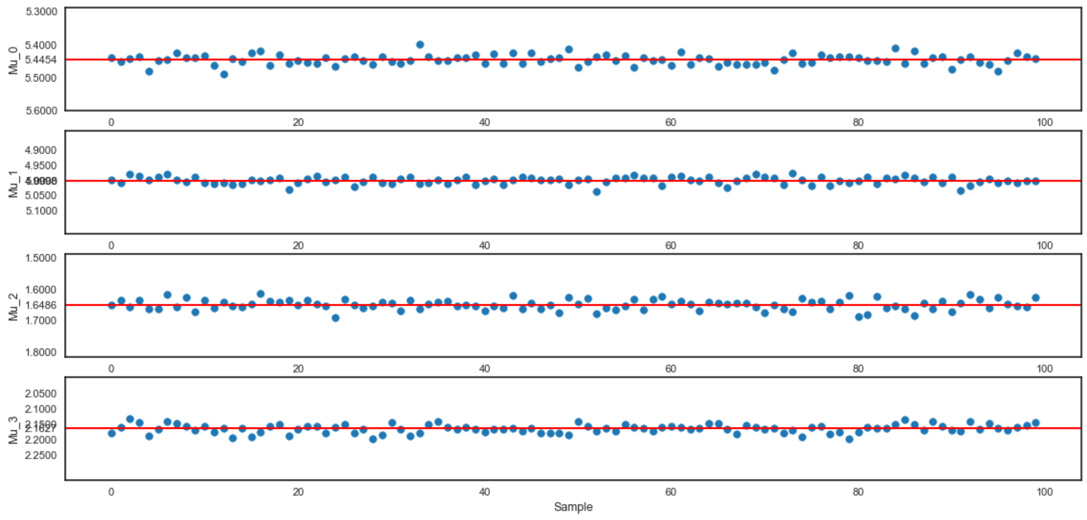

Figure 6.13: 4D Clustering problem: Mean values of cluster K_3sampled 100 times

μ_0	μ_1
3.0	2.0
-2.8	3.13
-3.78	-3.28
3.58	-1.85

μ_0	μ_1
-2.82373	2.64739
2.97223	-1.21415

Table 6.2: DPMM with REPS and $\alpha_0 = 10$: Comparison of the zeros of the Himmelblau function with means of the final sub-policies

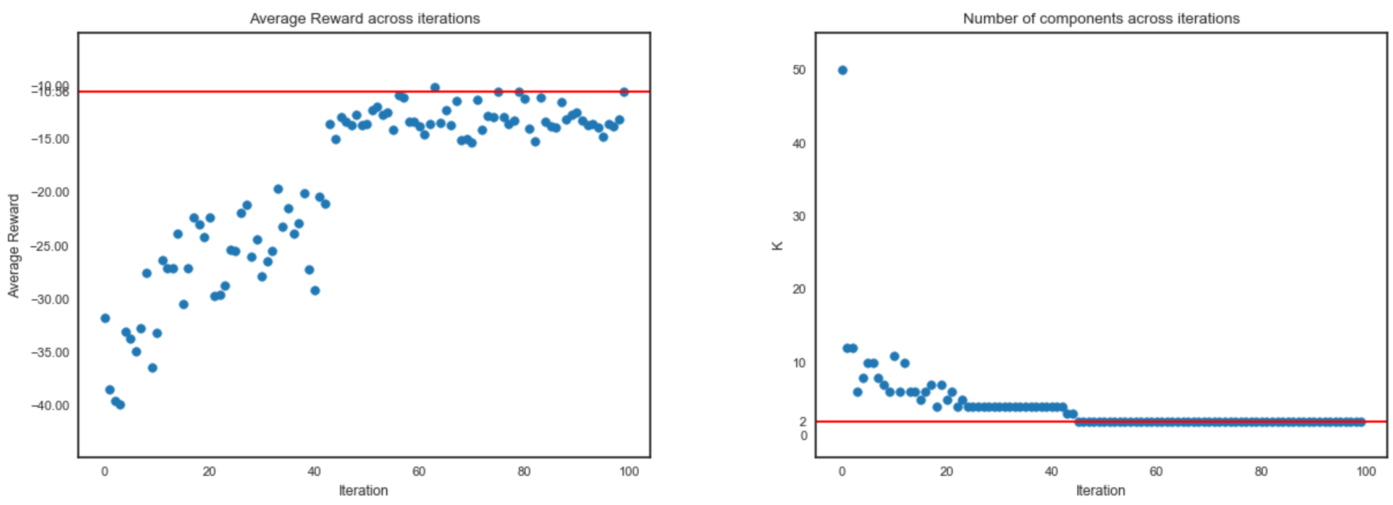

Figure 6.14: DPMM with REPS and $\alpha_0 = 10$: average reward and number of mixture components across iterations of the Gibbs sampler

6.3 Dirichlet Process Mixture Model with REPS

6.3.1 First Run

At the first run with concentration parameter $\alpha_0 = 10$, a comparison of the final optimized sub-policies means with the actual zeros of the Himmelblau function can be seen in Table 6.2.
In Figure 6.14, it is possible to see the total average reward of all sub-policies and the total number of components across the iterations of the Gibbs sampler.

6.3.2 Second Run

At the second run with concentration parameter $\alpha_0 = 15$, a comparison of the final optimized sub-policies means with the actual zeros of the Himmelblau function can be seen in Table 6.3.
In Figure 6.15, it is possible to see the total average reward of all sub-policies and the total number of components across the iterations of the Gibbs sampler.

6.3.3 Third Run

At the third run with concentration parameter $\alpha_0 = 11$, a comparison of the final optimized sub-policies means with the actual zeros of the Himmelblau function can be seen in Table 6.4.
In Figure 6.16, it is possible to see the total average reward of all sub-policies and the total number of components across the iterations of the Gibbs sampler.

μ_0	μ_1
3.0	2.0
-2.8	3.13
-3.78	-3.28
3.58	-1.85

μ_0	μ_1
3.3643	1.90181
-3.98164	2.83827
-3.49055	-3.43048

Table 6.3: DPMM with REPS and $\alpha_0 = 15$: Comparison of the zeros of the Himmelblau function with means of the final sub-policies

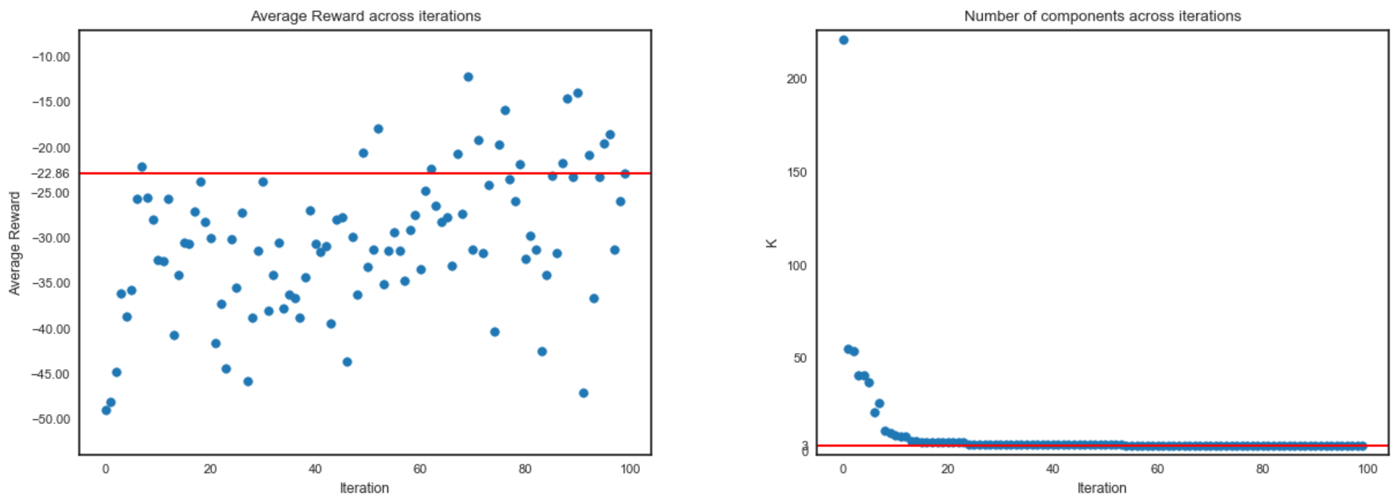

Figure 6.15: DPMM with REPS and $\alpha_0 = 15$: average reward and number of mixture components across iterations of the Gibbs sampler

μ_0	μ_1
3.0	2.0
-2.8	3.13
-3.78	-3.28
3.58	-1.85

μ_0	μ_1
-2.71741	-2.52
-3.80124	-3.46003
3.46121	-1.42731

Table 6.4: DPMM with REPS and $\alpha_0 = 11$: Comparison of the zeros of the Himmelblau function with means of the final sub-policies

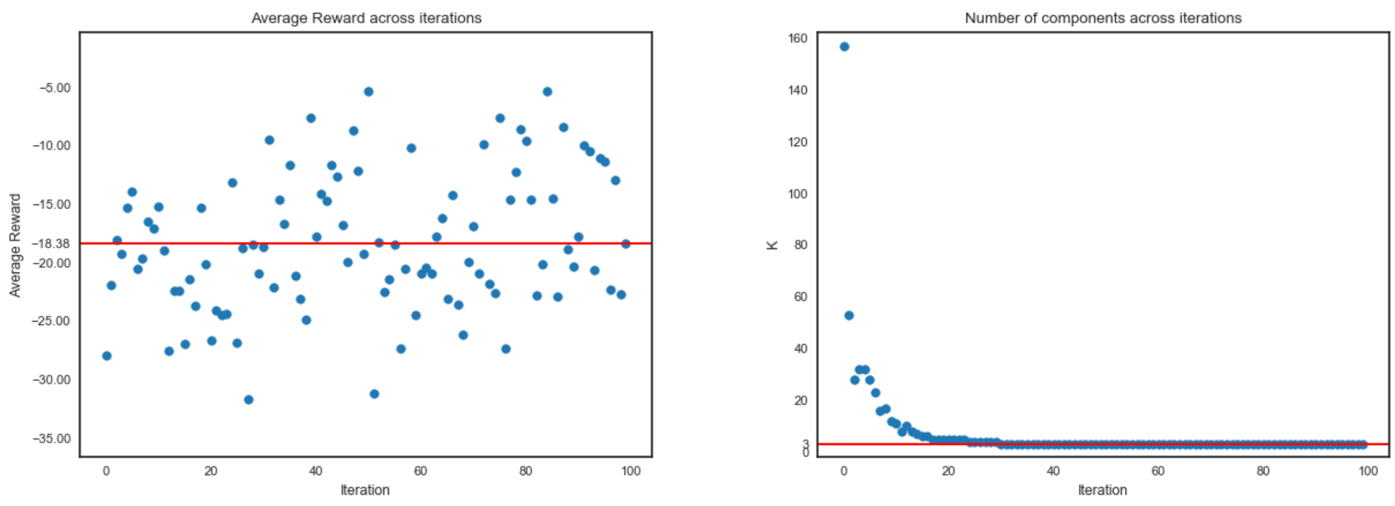

Figure 6.16: DPMM with REPS and $\alpha_0 = 11$: average reward and number of mixture components across iterations of the Gibbs sampler

7 Discussion

7.1 Learning Performance

Comparing the performance of the Gibbs sampler in the 2D and 4D environment (Figures 6.2 and 6.9), it is possible to observe how harder it was for the Gibbs sampler to cluster the data points accurately for a higher-dimensional problem, along with ending the iterations with a higher number of final clusters. This might indicate a possible difficulty in applying this method to a high-dimensional robotic set-up.

The trials of the Dirichlet Process Mixture Model with REPS showed a lot of difficulties in achieving a satisfactory result. It wasn't possible to find all of the four solutions in any of the presented tests, while the solutions that were found presented a low precision, which may lead to bad final results as well as a very unstable learning (Figures 6.15 and 6.16) process depending on the reward function under optimization. The results also show a quick convergence in all cases, even though not all of the solutions were found or the precision of the present solutions is not that high.

The lack of robustness shown by the presented results indicates a high sensitivity of the algorithm to the hyperparameters of both upper and lower level policies. The usage of an algorithm such as REPS to optimize the sub-policies requires a lot of available samples in order to present good results, but unfortunately the computational cost of the Gibbs sampler is too high, making the increase of data samples prohibitive.

7.2 Conclusion

The presented results show that it is possible to obtain multiple solutions to a task with a hierarchical representation using infinite mixture models such as Dirichlet Process. However, it is still an open problem how to guarantee the convergence to all of the solutions instead of only part of them. In HiREPS, this problem was mitigated with the introduction of a limit to the expected change in the entropy H of the responsibilities of the sub-policies. As these responsibilities are directly correlated to the concentration parameter α_0, it is reasonable to assume that developing a proper method for initializing and updating α_0 might be a way to achieve a satisfactory non-parametric hierarchical policy search method.

8 Outlook

In order to use a higher number of samples to improve the quality of REPS's output, it is possible to replace the Gibbs sampler by a much less computational-costly sampling method to fit the new parameters of the policy, such as the Variational Inference method [8].

Another open problem is how to properly control the learning in the upper-layer policy with the concentration parameter α_0 in order to keep a higher number of active components during the learning phase and not to loose important information in the process. Layered approaches [5] in policy search explore this issue and should be further investigated to improve the model.

Bibliography

[1] J. Peters, K. Muelling, and Y. Altun, “Relative entropy policy search,” vol. 2012.

[2] A. Abdolmaleki, J. Peters, and G. Neumann, “Model-based relative entropy stochastic search,”

[3] A. Prékopa, *Stochastic programming*. 1995.

[4] C. Daniel, O. Kroemer, G. Neumann, and J. Peters, “Hierarchical relative entropy policy search,” 2000.

[5] F. End, R. Akrour, J. Peters, and G. Neumann, “Layered direct policy search for learning hierarchical skills,” in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6442–6448, IEEE, 29/05/2017 - 03/06/2017.

[6] C. E. Rasmussen, “The infinite gaussian mixture model,” vol. 2000.

[7] R. M. Neal, “Markov chain sampling methods for dirichlet process mixture models,” 1988.

[8] D. M. Blei and M. I. Jordan, “Variational inference for dirichlet process mixtures,” 2006.

[9] Marc Peter Deisenroth, Gerhard Neumann and Jan Peters, “A survey on policy search for robotics.”

[10] X. Yu, “Gibbs sampling methods for dirichlet process,” vol. 2009.

[11] *2017 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 29/05/2017 - 03/06/2017.

[12] K. Murphy, *Machine Learning: A Probabilistic Perspective*. 2012.

[13] M. D. Escobar and M. West, “Bayesian density estimation and inference using mixtures,” 1995.