Multiagent Planning with Bayesian Nonparametric Asymptotics

by

Trevor D. J. Campbell

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of

Master of Science in Aeronautics and Astronautics

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

August 2013
[SEPTEMBER 2013]

Author .....
Department of Aeronautics and Astronautics
August 22, 2013

Certified by .....
Jonathan P. How
Richard C. Maclaurin Professor of Aeronautics and Astronautics
Thesis Supervisor

Accepted by .....
Professor Eytan H. Modiano
Professor of Aeronautics and Astronautics
Chair, Graduate Program Committee

Multiagent Planning with Bayesian Nonparametric Asymptotics

by

Trevor D. J. Campbell

Submitted to the Department of Aeronautics and Astronautics
on August 22, 2013, in partial fulfillment of the
requirements for the degree of
Master of Science in Aeronautics and Astronautics

Abstract

Autonomous multiagent systems are beginning to see use in complex, changing environments that cannot be completely specified a priori. In order to be adaptive to these environments and avoid the fragility associated with making too many a priori assumptions, autonomous systems must incorporate some form of learning. However, learning techniques themselves often require structural assumptions to be made about the environment in which a system acts. Bayesian nonparametrics, on the other hand, possess structural flexibility beyond the capabilities of past parametric techniques commonly used in planning systems. This extra flexibility comes at the cost of increased computational cost, which has prevented the widespread use of Bayesian nonparametrics in realtime autonomous planning systems.

This thesis provides a suite of algorithms for tractable, realtime, multiagent planning under uncertainty using Bayesian nonparametrics. The first contribution is a multiagent task allocation framework for tasks specified as Markov decision processes. This framework extends past work in multiagent allocation under uncertainty by allowing exact distribution propagation instead of sampling, and provides an analytic solution time/quality tradeoff for system designers. The second contribution is the Dynamic Means algorithm, a novel clustering method based upon Bayesian nonparametrics for realtime, lifelong learning on batch-sequential data containing temporally evolving clusters. The relationship with previous clustering models yields a modelling scheme that is as fast as typical classical clustering approaches while possessing the flexibility and representational power of Bayesian nonparametrics. The final contribution is Simultaneous Clustering on Representation Expansion (SCORE), which is a tractable model-based reinforcement learning algorithm for multimodel planning problems, and serves as a link between the aforementioned task allocation framework and the Dynamic Means algorithm.

Thesis Supervisor: Jonathan P. How
Title: Richard C. Maclaurin Professor of Aeronautics and Astronautics

Acknowledgments

There are a number of people who have my utmost appreciation for helping me along my travels here at MIT. First and foremost, my advisor Jon – it has been, and will continue to be, a pleasure exploring and developing the world of autonomous planning with you. Your supervision has been crucial in shaping my research interests, how I approach problems, and how I communicate my findings to the academic world.

In no uncertain terms do I owe my success in my endeavours to my friends and family. Mom, Dad, Emily – thank you for keeping me sane, giving me a brief respite from the fast-paced world of research from time to time, and celebrating with me when I made breakthroughs. To my group of awesome friends from the University of Toronto - Sean, Rick, Manan, Jamie, Konstantine, Adam, Sanae, Catherine, Amy, Angela, and countless more – thanks for sticking with me through thick and thin. To my new friends here at MIT – Sam, Luke, Buddy, Ian, Dan, Andrew, Kemal, Bobby, Chris, Jack, Rob, Kane, Miao, Andrew, Nikhil, Vyas, and everyone else – thanks for making me feel right at home here in Boston. To Brendan, I've had the time of my life being a student of yours – you've opened my mind to a completely new world of improvisation, voicings, and keyboard geometry that, until meeting you, I had no idea existed.

Last, but most certainly not least, I thank my wonderful girlfriend Maria. Your love and support have been the foundation upon which I have stood, and your perseverence in your own academics is an inspiration to me.

This work was supported by ONR MURI Grant N000141110688, and a Natural Sciences and Engineering Research Council of Canada PGS-M grant.

For fifteen days I struggled to prove that no functions analogous to those I have since called Fuchsian functions could exist. I was then very ignorant; every day I sat down at my work table where I spent an hour or two, tried a great number of combinations and arrived at no result. One evening, contrary to my custom, I took black coffee. I could not go to sleep; ideas swarmed up in clouds, and I sensed them clashing until, to put it so, a pair would hook together to form a stable combination. By morning I had established the existence of a class of Fuchsian functions, those derived from the hypergeometric series. I had only to write up the results, which took but a few hours.

Henri Poincaré

Contents

Chapter 1

Introduction

1.1 Overview

Autonomous multiagent systems are becoming increasingly prevalent in situations involving complex, changing environments that cannot be completely specified a priori [1–5]. In such situations, the system must coordinate and make decisions based upon information gathered from the environment in which it acts. There are a number of ways to take observed data into account when planning; one of the most popular techniques is a *model-based* approach, where observations are used to update a compact, approximate representation (a *model*) of the environment, and decisions are made with respect to that model. Having an accurate model is often paramount to the success of the autonomous system; however, in realistic missions, an accurate model generally cannot be obtained with certainty. The observations made by the system are often noisy, incomplete, and local, leading to uncertainty in which model best captures the relevant aspects of the environment.

A principled approach of dealing with this uncertainty is to use a probabilistic model of the environment. Such techniques involve specifying an initial model, which is then updated using Bayesian posterior inference as observations are made [6–8]. This initial model has, in the vast majority of probabilistic planning literature, fallen into the class of *parametric* probabilistic models (e.g. the beta-Bernoulli model [9]). The use of such parametric models requires the system designer to take a certain leap

of faith; parametric models assume the structure of the model, or its number of constituent components, is well-known and fixed. Of course, both of these assumptions are routinely violated by real-world planning missions; the model structure is rarely known a priori, and even more rarely is it constant throughout the duration of the mission.

Bayesian nonparametric models (BNPs), on the other hand, are a class of probabilistic models in which the number of parameters is mutable, and may be modified during posterior inference [10–15]. BNPs have a flexible structure and, therefore, are particularly useful in situations where it is not clear what the model structure should be a priori. The designer is thus free from specifying the model structure, instead allowing the system to discover it through interactions with its environment. However, the enhanced flexibility of Bayesian nonparametrics with respect to their parametric counterparts does not come without a cost. Current inference techniques, such as Gibbs sampling [16], variational inference [17], stochastic variational inference [18], and particle learning [19], are not computationally competitive with parametric inference techniques.

This problem of computational tractability is not unique to the learning component of a model-based autonomous system; past approaches to multiagent planning under uncertainty suffer from similar pitfalls. Markov decision processes (MDPs) are a natural framework for describing sequential decision-making problems, can capture rich forms of uncertainty in the evolution of the surrounding environment, and have been used in a wide variety of applications [20]. However, they suffer from the "curse of dimensionality": the difficulty of solving the planning problem scales exponentially with the number of agents in the system. Recent approaches have mitigated this to an extent, but are still computationally expensive and sacrifice explicit, guaranteed coordination [21]. Multiagent task allocation approaches, on the other hand, have a computational cost that scales polynomially in the number of agents, and guarantee explicit coordination; however, to date, all such methods that account for uncertainty do not perform model learning based on observations, and require computationally expensive sampling procedures to evaluate the performance of candidate allocations

[22]. The issues of computational cost (both incurred by the learning and planning procedures) are particularly relevant when considering the computational power available in current embedded systems, such as those found in many autonomous multiagent systems.

Thus, the goal of this thesis is the development of a Bayesian nonparametric model-based multiagent planning system, with a focus on the tractable incorporation of uncertainty at both the planning and learning stages.

1.2 Literature Review and Analysis

Multiagent Task Allocation Task allocation algorithms generally solve the problem of deciding, given a list of tasks to do and available resources, an assignment of resources to tasks that maximizes some notion of overall system performance [23]. The problem statement most commonly used is that of a general mixed-integer, nonlinear optimization program. As this mathematical program is intractable to solve exactly, solution techniques generally involve heuristics (e.g. sequential greedy allocation). Recent advances in this literature have provided polynomial-time, asynchronous, decentralized auctions with guaranteed convergence [24–28], but such advances are designed to provide assignments when task models have a constant, deterministic state (e.g. visiting a set of waypoints). These greedy assignment algorithms have, however, been extended to include the effects of stochastic parameters on the overall task assignment process, and have considered robust, chance-constrained, and expected performance metrics [22, 29]. The most successful of these were the chance-constrained approaches; however, they rely on sampling a set of particles from the distributions of stochastic parameters and making informed decisions based on those particles, with no notion of solution quality vs. the number of propagated samples. Further, these approaches do not incorporate learning to improve the stochastic models over time. Conversely, there are approaches which do incorporate learning (e.g. the Intelligent Cooperative Control Architecture (iCCA) [30, 31]) to reduce model uncertainty, but plan using the expected model and do not actually account for the uncertainty in the model during

the allocation procedure itself. Thus, while scalable coordination under uncertainty has been studied within this framework, the areas of concurrent learning and allocation under uncertainty, and exact evaluation of assignment score under uncertainty remain un-addressed.

Markov Decision Processes In the framework of Markov decision processes, explicit and guaranteed multiagent coordination for general problems is well known to exhibit exponential computational scaling with respect to the number of agents in the team [21]. On the other hand, a number of previous studies have explored the benefits of decomposing MDPs into smaller *weakly coupled* [32] concurrent processes, solving each individually, and merging the solutions to form a solution of the larger problem [33, 34]. Building on that work, several researchers have considered problems where shared resources must be allocated amongst the concurrent MDPs [32, 35]; generally, in such problems, the allocation of resources amongst the MDPs determines which actions are available in each MDP, or the transition probabilities in each MDP. Some approaches involve solving resource allocation or coordination problems by modelling them within the MDP itself [36, 37]. Assignment based decompositions have also been studied [38], where cooperation between agents is achieved via coordinated reinforcement learning [39]. However, none of these approaches have considered the impact of previous tasks on future tasks. This is of importance in multiagent multi-assignment planning problems; for instance, if an agent is assigned a task involving a high amount of uncertainty, that uncertainty should be properly accounted for in all of its subsequent tasks (whose starting conditions depend on the final state of the uncertain task). This allows the team to hedge against the risk associated with the uncertain task, and allocate the subsequent tasks accordingly. Past work has focused primarily on parallel, infinite-horizon tasks [32, 35], and have not encountered the cascading uncertainty that is characteristic of sequential bundles of uncertain tasks.

Bayesian Nonparametrics – Learning Several authors have explored BNP techniques within the context of machine learning [10, 11, 13, 15, 40–46]. For example,

BNPs have been used for speaker diarization [14], for document classification [11], for classifying life history data [46], and for classifying underwater habitats [47]. Many Bayesian nonparametric models exist, such as the Dirichlet process and its hierarchical variant [10, 11], the beta process and its hierarchical variant [13], and the dependent Dirichlet process [12, 48]. The Dirichlet and dependent Dirichlet processes are the primary BNPs used in this thesis; the reader is encouraged to consult the references for more complete discussions of the other existing models.

The Dirichlet process mixture model (DPMM) is a powerful tool for clustering data that enables the inference of an unbounded number of mixture components, and has been widely studied in the machine learning and statistics communities [17, 19, 49, 50]. Despite its flexibility, it assumes the observations are exchangeable, and therefore that the data points have no inherent ordering that influences their labeling. This assumption is invalid for modeling temporally/spatially evolving phenomena, in which the order of the data points plays a principal role in creating meaningful clusters. The dependent Dirichlet process (DDP), originally formulated by MacEachern [48], provides a prior over such evolving mixture models, and is a promising tool for incrementally monitoring the dynamic evolution of the cluster structure within a dataset. More recently, a construction of the DDP built upon completely random measures [12] led to the development of the dependent Dirichlet process Mixture model (DDPMM) and a corresponding approximate posterior inference Gibbs sampling algorithm. This model generalizes the DPMM by including birth, death and transition processes for the clusters in the model.

While Bayesian nonparametrics are powerful in their capability to capture complex structures in data without requiring explicit model selection, they suffer some practical shortcomings. Inference techniques for BNPs typically fall into two classes: sampling methods (e.g. Gibbs sampling [50] or particle learning [19, 51]) and optimization methods (e.g. variational inference [17] or stochastic variational inference [18]). Current methods based on sampling do not scale well with the size of the dataset [52]. Most optimization methods require analytic derivatives and the selection of an upper bound on the number of clusters a priori, where the computational complexity in-

creases with that upper bound [17, 18]. State-of-the-art techniques in both classes are not ideal for use in contexts where performing inference quickly and reliably on large volumes of streaming data is crucial for timely decision-making, such as autonomous robotic systems [53–55]. On the other hand, many classical clustering methods [56–58] scale well with the size of the dataset and are easy to implement, and advances have recently been made to capture the flexibility of Bayesian nonparametrics in such approaches [57–60]. There are also a number of sequential Monte-Carlo methods that have capabilities similar to the DDP mixture model [61, 62] (of which particle learning may be seen as a generalization). However, as of yet, there is no algorithm that captures dynamic cluster structure with the same representational power as the DDP mixture model while having the low computational workload of classical clustering algorithms.

Bayesian Nonparametrics – Planning Within the context of planning, BNPs are just beginning to make a substantial impact. The robotics community has perhaps made the most use of BNPs so far; however, the attention here has been limited primarily to Gaussian processes (GPs). GP-based models have been used for regression and classification [15], for Kalman filtering [63], and for multistep lookahead predictions [64]. Several authors have also used GPs for planning in the framework of dynamic programming and reinforcement learning [65, 66], and in motion planning in the presence of dynamic, uncertain obstacles [67]. The Dirichlet process (DP) has been combined with the GP to form the DPGP mixture model, which has seen use in target tracking [68]. Finally, hierarchical Dirichlet processes have seen use in various Markov systems, such as hidden Markov models [14] and partially observable Markov decision processes [69]. This lack of substantial BNP research in planning has been noted in previous studies [70].

Multiple Model-based Reinforcement Learning This thesis employs model-based reinforcement learning (RL) to capture systems that exhibit an underlying multiple-model (or *multimodel*) structure, i.e. those that can be described as a col-

lection of distinct models with some underlying mechanism for switching between them. Generally speaking, the model-based RL paradigm involves building an accurate model of a MDP from observed interactions [71, 72]. There is a vast body of prior literature on model-based reinforcement learning methods, such as classical methods (e.g. Dyna [71, 73, 74]), Bayesian methods (e.g. BOSS [75], BEETLE [76]), methods with statistical error bounds (e.g. R-Max [77], KWIK [78]), among others. However, these techniques do not incorporate the aforementioned multimodel structure, resulting in a learned model that is the average over the set of underlying models. Early work on controlling systems with multiple models that does take advantage of the structure, such as MOSAIC [79] and MPFIM [80], used a softmax mixture of control experts where weights are determined by predictive likelihood. Multiple model-based reinforcement learning (MMRL) [81], built upon these methods, involves using a mixture of experts for the model representation, where the weighted experts are used for updating value functions, models, and deciding which action to take. MOSAIC-MR [82] extends MMRL and MOSAIC to the case where changes in the reward function are unobservable, and does not pair controllers a priori with fixed predictors of dynamics and reward. One major drawback of approaches such as MOSAIC-MR is that the number of models is fixed and known a priori, as is the chosen representation; this limits the capability of such approaches to handle situations where the number of models is unknown a priori, in addition to their individual complexity. Outside of the domain of reinforcement learning, a number of Bayesian nonparametric methods exist that learn an unknown number of underlying models within a single system. For example, the sticky HDP-SLDS-HMM and HDP-AR-HMM [83] learn a set of linear or autoregressive dynamical systems, where the switching between the models is controlled by a hidden Markov model with an unknown number of states. However, the structure of each underlying model is fixed and does not adapt to better capture the data. The DPGP [68] is a good example of a BNP that learns an underlying multiple model structure; the number of models is unknown, and each Gaussian process (GP) model adapts to fit the data assigned to it. However, using the original DPGP (and indeed, most Bayesian nonparametric inference techniques [17–19]) in an RL setting

is impractical due to the complexity of inference.

1.3 Thesis Contributions and Organization

The rest of this thesis proceeds as follows. Chapter 2 provides a technical background in Bayesian nonparametric modelling, Markov decision processes, and multiagent task allocation. Chapters 3, 4, and 5 highlight the three major contributions of this thesis:

- Chapter 3 presents the development of a multiagent task allocation framework for tasks specified as Markov decision processes. This framework extends past work in multiagent allocation under uncertainty by incorporating the generality of Markov decision processes, allows exact distribution propagation instead of sampling, and provides an analytic solution time/quality tradeoff for system designers. Empirical results corroborate the theoretical results, demonstrating that this method remains tractable in the presence of a large number of agents and uncertain tasks. The discussion in this chapter makes the assumption that all system models are well-known.

- Chapter 4 introduces the tool that will serve to relax the assumption that the system models are well known: the Dynamic Means algorithm, a novel clustering method based upon Bayesian nonparametrics for real-time, lifelong learning on batch-sequential data containing temporally evolving clusters. This algorithm is developed by analyzing the low-variance asymptotics of the dependent Dirichlet process mixture model, and is guaranteed to converge in a finite number of iterations to a local optimum in a k-means-like cost function. Empirical results demonstrate that the Dynamic Means algorithm is faster than typical classical hard and probabilistic clustering approaches, while possessing the flexibility and representational power of Bayesian nonparametrics.

- Chapter 5 discusses a reinforcement learning framework, the Simultaneous Clustering on Representation Expansion (SCORE) algorithm, that links the concepts in Chapters 3 and 4. By combining the strengths of representation expan-

sion and hard clustering, this algorithm provides a method for fast, model-based reinforcement learning when the planning problem is large and exhibits an underlying multiple-model structure. The algorithm is guaranteed to terminate in a finite number of iterations to a local optimum in a combined clustering-expansion cost function. Empirical results on a simulated domain demonstrate that this method remains tractable for large state-spaces and outperforms contemporary techniques in terms of both the sample and time complexity of learning.

The three aforementioned contributions together form a full toolset for combined planning and learning, which operates in real-time and has the flexibility of Bayesian nonparametrics.

Finally, Chapter 6 concludes the thesis and provides future directions for further investigation.

Chapter 2

Background

2.1 Overview

This chapter provides a comprehensive review of the background material required to understand this thesis. It covers the basics of Bayesian nonparametric modeling and inference (and in particular, the Dirichlet and dependent Dirichlet processes), Markov decision processes, and multiagent task allocation.

2.2 Bayesian Nonparametrics

Bayesian modelling and inference generally involves using measurements to improve a probabilistic model of an observed phenomena. The two major components of a Bayesian probabilistic model are the *likelihood* and the *prior* distributions: The likelihood $p(y|\theta)$ is a distribution that represents the probability of the observed data y given a set of parameters θ, while the prior $p(\theta)$ represents the probability distribution over those parameters (sometimes referred to as latent variables) before any observations are made. As measurements become available, the prior is combined with the likelihood of those measurements using Bayes' rule, resulting in a *posterior* distribution $p(\theta|y)$ over the parameters θ:

$$p(\theta|y) = \frac{p(y|\theta)p(\theta)}{p(y)} \tag{2.1}$$

Where the posterior may be thought of as an updated distribution over θ that takes into account all information contained in the observations y.

If the posterior $p(\theta|y)$ is of the same family of distributions as the prior $p(\theta)$, when using a particular likelihood $p(y|\theta)$, the prior is said to be *conjugate* to that likelihood. For example, if $y|\theta \sim \mathcal{N}(\theta, \sigma^2)$, then $\theta \sim \mathcal{N}(\mu, \tau^2)$ is a conjugate prior that results in the posterior distribution $\theta|y \sim \mathcal{N}\left(\frac{\mu}{\tau^2} + \frac{y}{\sigma^2}, \frac{1}{\frac{1}{\sigma^2} + \frac{1}{\tau^2}}\right)$, where $\mathcal{N}(a, b)$ is a normal distribution with mean a and variance b. In other words, a normal prior is conjugate to a normal likelihood. The use of conjugate priors leads to an ability to perform closed-form, exact posterior inference, rather than relying on approximate methods; in this case, the *hyperparameters* (parameters of the prior $p(\theta)$), μ and τ, can be updated analytically to $\mu' = \frac{\mu}{\tau^2} + \frac{y}{\sigma^2}$ and $\tau' = \frac{1}{\frac{1}{\sigma^2} + \frac{1}{\tau^2}}$. This capability becomes very important when dealing with complex Bayesian models, where the closed-form conjugate updates constitute part of a larger inference algorithm [16].

Typically, a Bayesian model is specified in terms of a *generative model*, which is a conceptual process by which the observations are assumed to be created. The generative model defines the likelihood(s) and prior(s) for use in inference. As a brief illustrative example, say there is a database of N images y_i of K objects, where the object in each image is unlabelled. In order to label the images in terms of the object they contain, the images must be grouped into K bins, one for each object. An appropriate generative model for this situation might be as follows:

1. For each bin $k = 1, \ldots, K$, sample a set of parameters $\theta_k \sim H$
2. For each $i = 1, \ldots, N$, roll a weighted K-sided die and observe the outcome z_i
3. For each $i = 1, \ldots, N$, sample the image $y_i \sim \mathcal{N}(\theta_{z_i}, \sigma^2)$

If written mathematically, this can be expressed as:

$$\begin{aligned} \pi|\alpha &\sim \text{Dir}(\alpha_1, ..., \alpha_K) \\ z_i|\pi &\sim \text{Categorical}(\pi),\ i = 1, \ldots, N \\ \theta_k &\sim \mathcal{N}(\mu_*, \sigma_*^2)\ k = 1, \ldots, K \\ y_i|z_i, \theta &\sim \mathcal{N}(\theta_{z_i}, \sigma^2). \end{aligned} \tag{2.2}$$

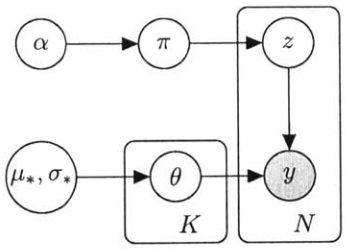


Figure 2-1: Dirichlet mixture graphical model

where the latent variables of the generative model are as follows:

- $\pi = (\pi_1, \ldots, \pi_K)$, $\sum_k \pi_k = 1$ are a set of positive weights that sum to 1 (the probabilities of the weighted K-sided die),
- $\{z_i\}_{i=1}^N$, $z_i \in \{1, \ldots, K\}$ are the bin index labels for each image,
- $\theta = (\theta_1, \ldots, \theta_K)$ are the parameters that govern the appearance of each object.

The hyperparameters of the prior are $\alpha_1, \ldots, \alpha_K > 0$, and represent imaginary counts of rolls from the weighted K-sided die before the experiment begins. For example, setting $\alpha_1 = 100$, and $\alpha_k = 0.1$, $k = 2, \ldots, K$ creates a high prior certainty on side 1 of the die being heavily weighted (as if a die were rolled 100 times and landed on the 1 side every time), while setting $\alpha_k = 100, k = 1, \ldots, K$ creates a high prior certainty that the die is fair (as if a die were rolled $100K$ times, and landed on each side 100 times). The α_k may be selected by hand, in which case a larger set of $\alpha_1, \ldots, \alpha_K > 0$ yields a stronger prior that requires more data to overcome, and vice versa for smaller $\alpha_1, \ldots, \alpha_K > 0$.

This particular generative model has a name: the Dirichlet *mixture model* [84]. Of course, this is not how the images were actually created; it is simply a conceptual process in order to derive appropriate likelihood and prior distributions over the data and latent variables. Here, the prior distributions are the Dirichlet-Categorical distribution over π and z_i, and the multivariate Gaussian distribution over θ; and the likelihood is the Gaussian distribution over y_i. As measurements become available, the estimates of π and θ can be improved via Bayesian inference.

Equation (2.2) is an example of a *parametric* Bayesian model. In such models,

the number of parameters or latent variables in the model are fixed a priori (in this case, the K mixing probabilities and parameters), and are often chosen through expert judgment. Bayesian nonparametric models, in contrast, do not require the expert specification of the number of parameters; instead, the number of parameters in such models is infinite, while a finite quantity of observed data is assumed to be generated using only a finite subset of those parameters. This allows Bayesian inference techniques to learn the *number* of latent variables in a model, in addition to their values.

Table 2.1 presents a list of standard and recently developed Bayesian nonparametric models [85]. The Gaussian process (GP) is perhaps the most well known example of a BNP in the planning community, and is typically used as a model for continuous functions. GP-based models have been used for regression and classification [15], for Kalman filtering [63], and for multistep lookahead predictions [64]. Several authors have also used GPs for planning in the framework of dynamic programming and reinforcement learning [65, 66], in motion planning in the presence of dynamic, uncertain obstacles [67], and in predicting pedestrian motion [86]. The Dirichlet process (DP) and beta process (BP), on the other hand, are well-known nonparametric models in the machine learning community [10, 14, 17, 85], but have seen very few applications in planning and control, despite their ability to robustly capture the complexity of a system solely from observations [68, 87]. The DP and BP also serve as building blocks for other Bayesian nonparametric priors, such as the dependent Dirichlet process (DDP) [12], the hierarchical Dirichlet process (HDP) [11], and the hierarchical Beta process (HBP) [13].

This thesis is focused on the applications and advantages of the DDP and models based thereupon in planning systems. As such, the DP and DDP will be explained in detail in the following; citations for the other common Bayesian nonparametrics may be found in the list of references.

Table 2.1: Typical BNPs with applications [85]

Model	Typical Application
GP	Learning continuous functions
DP	Data clustering, unknown # of static clusters
BP	Latent feature models, unknown # of features
DDP	Data clustering, unknown # of dynamic clusters
HDP	Topic modeling, unknown # topics, words
HBP	Shared latent feature models, unknown # features

2.2.1 The Dirichlet Process

Recall that the Dirichlet distribution in the earlier mixture model example (2.2) is a prior over unfair K-sided dice, and is used in mixture models with K components. One might naturally wonder whether the mixture model can be relaxed, such that K can be learned from the data itself without expert specification. As mentioned in the previous section, this might be accomplished by assuming that $K \to \infty$ (in a sense), and then taking advantage of the fact that a finite quantity of data must have been generated by a finite number of components.

Proceeding based on this notion, let the k- dimensional unit simplex be defined as the set of vectors $\pi = (\pi_1, \ldots, \pi_K) \in \mathbb{R}^K$ such that $\sum_{i=1}^{K} \pi_i = 1$, and $\pi_i \geq 0$ for all i. Then, the Dirichlet distribution is a probability distribution over this K-dimensional unit simplex whose density is given by:

$$p(\pi) = \frac{\Gamma(\sum_{i=1}^{K} \alpha_i)}{\sum_{i=1}^{K} \Gamma(\alpha_i)} \prod_{i=1}^{K} (\pi_i)^{\alpha_i - 1}, \tag{2.3}$$

where $\Gamma(\cdot)$ is the Gamma function, and $\alpha = (\alpha_1, ..., \alpha_K)$ is a parameter vector. If one sets $(\alpha_1, ..., \alpha_K) = \left(\frac{a}{K}, ..., \frac{a}{K}\right)$ for some $a > 0$, taking the limit as $K \to \infty$ creates a Dirichlet distribution whose draws contain an infinite number of components that sum to 1. This is the intuitive underpinning of the Dirichlet process (DP). The definition of a Dirichlet process is as follows (developed formally in [44]):

Definition 2.2.1 *[44] Let H be a probability measure on a space X, and let $\alpha > 0$ be a real number. Then $G \sim DP(\alpha, H)$ is a probability measure drawn from a Dirichlet process if for any finite partition $\{B_1, ..., B_N\}$ of X (i.e. $B_i \bigcap B_j = \emptyset$ for all $i \neq j$*

and $\bigcup_{i=1}^{N} B_i = X$*),*

$$(G(B_1), ..., G(B_N))|H, \alpha \sim \text{Dir}(\alpha H(B_1), ..., \alpha H(B_N)). \tag{2.4}$$

This definition illustrates the fundamental difference between the conceptual "infinite Dirichlet distribution" described earlier and a Dirichlet process: samples from the former are ∞-dimensional vectors, while samples from the latter are *probability measures* on some space X. Further, it can be shown that $E[G(B)] = H(B)$, and that $V[G(B)] = H(B)(1 - H(B))/(\alpha + 1)$ for any $B \subset X$ [10]. This provides a sense of how H and α influence G: H, called the *base measure*, is essentially the mean of G; and α, called the *concentration parameter*, controls its variance. Occasionally, one may use the notation $G \sim \text{DP}(\mu)$, where μ is a measure on X that does not necessarily sum to 1. In this case, it is understood that $\alpha = \int_X \text{d}\mu$, $H = \mu/\alpha$.

The definition of the Dirichlet process is admittedly rather hard to draw any useful conclusion from at first glance. However, it can be shown that G is discrete with probability 1, and has the following form [88]:

$$G = \sum_{i=1}^{\infty} \pi_i \delta_{\theta_i}, \tag{2.5}$$

where the weights π_i satisfy $\sum_{i=1}^{\infty} \pi_i = 1$, δ_x is an atom at $x \in X$, and $\theta_i \sim H$ are the locations of the atoms. A representation of G is shown in Figure 2-2. This provides a mental picture of what a DP is: it can be thought of as a distribution, which, when sampled, yields another probability distribution G having a countably infinite number of atoms on a space X. Each atom has independent and identically distributed (i.i.d.) location within X with distribution H (with a slight abuse of notation , we use H to both refer to the measure and its related continuous distribution), where the weights π_i on the atoms can be thought of as being drawn from an infinite dimensional Dirichlet distribution.

In order to sample G from a DP, the *stick breaking construction* is typically used [14]. The stick breaking construction (sometimes denoted $\pi \sim \text{GEM}(\alpha)$) pro-

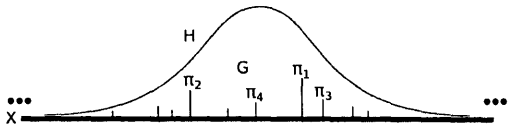


Figure 2-2: $G \sim \mathrm{DP}(\alpha, H)$.

Figure 2-3: Stick breaking procedure

vides an iterative procedure for sampling the weights of (2.5), given by:

$$\pi_i = \beta_i \prod_{j=1}^{i-1} (1 - \beta_j), \tag{2.6}$$

where $\beta_i \sim \mathrm{Beta}(1, \alpha)$ are sampled from the beta distribution. As shown in Fig. 2-3, this corresponds to taking a stick of unit length, repeatedly breaking off beta distributed pieces, and assigning them to locations θ_i in X. Thus, to sample a draw from a DP, we simply alternately sample θ_i from H, and calculate the corresponding π_i by sampling β_i. Of course, there are theoretically an infinite number of (π_i, θ_i) pairs, but the stick breaking process can be terminated to obtain an approximation after a finite number of steps by re-normalizing the weights to sum to 1, or by assigning whatever remaining probability mass there is to the last θ_i.

Posterior inference for the Dirichlet process prior shares strong connections to posterior inference for the Dirichlet distribution. Consider once again the parametric mixture model in (2.2); it was not stated earlier, but the Dirichlet distribution is the conjugate prior of the categorical (or more generally, multinomial) likelihood. In other words, if one wishes to learn the outcome probabilities on a K-sided unfair die, one may observe N outcomes z_i of the die, and then apply Bayes' rule:

$$\begin{aligned} \pi &\sim \mathrm{Dir}(\alpha_1, \ldots, \alpha_K) \\ \{z_i\}_{i=1}^N &\sim \mathrm{Multinomial}(\pi) \\ \therefore \pi | \{z_i\}_{i=1}^N &\sim \mathrm{Dir}(\alpha_1 + n_1, \ldots, \alpha_K + n_K) \end{aligned} \tag{2.7}$$

where $n_k = \sum_{i=1}^N [z_i = k]$, and the posterior $\pi | \{z_i\}_{i=1}^N$ is another Dirichlet distribution due to conjugacy.

The fact that samples from a Dirichlet process are similar to samples from an ∞-

dimensional Dirichlet distribution hints that the Dirichlet process is also conjugate to the categorical/multinomial likelihoods. Indeed this is the case; the posterior of a Dirichlet process after observing N samples θ_i drawn from $G \sim \text{DP}$ is

$$\begin{aligned} G &\sim \text{DP}(\alpha, H) \\ \{\theta_i\}_{i=1}^N &\sim G \\ G|\{\theta_i\}_{i=1}^N &\sim \text{DP}\left(\alpha,\ \frac{\alpha}{\alpha+N}H + \frac{1}{\alpha+N}\sum_{i=1}^{N}\delta_{\theta_i}\right) \end{aligned} \tag{2.8}$$

where the old base measure H is replaced by a weighted combination between H and the observed atoms δ_{θ_i}, with the weighting determined by α. However, as with the Dirichlet distribution, the Dirichlet process is most useful as a prior over mixture models where the θ_i themselves are not measured directly, but rather some noisy measurements y_i that depend on θ_i. Thus, while this description of the posterior Dirichlet process is illustrative mathematically, something more is required, practically speaking.

Towards this goal, a probabilistic model which is closely related (this relation will be elucidated shortly) to the Dirichlet process is the Chinese Restaurant process (CRP) [10]. This stochastic process models a sequence of data θ_i as being generated sequentially, and upon generation of each data point, it is assigned a random label z_i. Suppose a finite subset of such a sequence as been observed, and k unique labels have been assigned; then,

$$\begin{aligned} p(z_{n+1} = j|z_1, \ldots, z_n) &= \frac{n_j}{\alpha+n}, \quad \forall j \leq k \\ p(z_{n+1} = k+1|z_1, \ldots, z_n) &= \frac{\alpha}{\alpha+n} \end{aligned} \tag{2.9}$$

where $n_j = \sum_{i=1}^{n}[z_i = j]$. The name of the CRP comes from an analogy often used to describe it, which is illustrated in Fig. 2-4. If we consider a restaurant with an infinite line of tables, where customers enter and either sit at a previously selected table with probability proportional to the number of customers already sitting at that table, or start a new table with probability proportional to α, the table assignments are said

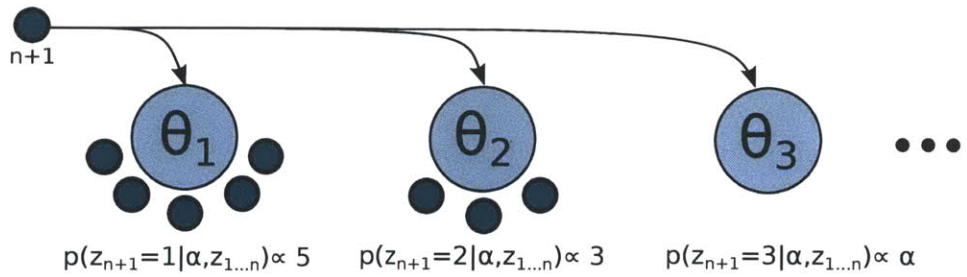


Figure 2-4: A pictorial representation of the CRP.

to be drawn from a Chinese Restaurant Process.

Perhaps the most useful fact about the CRP is that the labels z_i are *exchangeable*:

Definition 2.2.2 *A sequence of random variables $z_1, z_2, \ldots$ is exchangeable if for any finite collection $z_{i_1}, z_{i_2}, \ldots, z_{i_N}$,*

$$p(z_{i_1}, z_{i_2}, \ldots, z_{i_N}) = p(z_{k_1}, z_{k_2}, \ldots, z_{k_N}) \tag{2.10}$$

for all permutations $\{k_1, k_2, \ldots, k_N\}$ of $\{i_1, i_2, \ldots, i_N\}$.

The fact that the observations z_i in (2.9) are exchangeable is clear because the distributions only depend on the *number of times* previous labels have been observed, and not their order. The reason why this is so important is due to de Finetti's theorem [10]; informally, this theorem states that if a sequence of random variables is exchangeable, then there exists an *underlying distribution* which explains the data as if it were sampled i.i.d. from that distribution. Using de Finetti's theorem, the CRP and the DP can be shown to be equivalent – As each sequential data point is observed, it is impossible to tell whether it was sampled i.i.d. from the distribution $G \sim \mathrm{DP}(\alpha, H)$ or whether it was generated from the sequential cluster assignment rules of the CRP.

Thus, approximate posterior inference for the DP can be conducted using the distributions provided by the CRP. This will be discussed further in the following. The reader is encouraged to consult [10] for further discussion of the Dirichlet process.

2.2.2 The Dependent Dirichlet Process

The major strength of the Dirichlet process as a prior over mixture models is that it infers K, the number of components in the model, directly from the data in a Bayesian framework. While this level of functionality is sufficient for a wide variety of applications (from topic modeling [89] to trajectory pattern clustering [68]), in the context of planning it has a major weakness. The Dirichlet process is a static model that assumes that the data was generated from an unchanging set of mixture components. Autonomous planning systems often operate in dynamic environments where conditions are constantly changing, and even perhaps adapting to the system's behavior. Thus, it is important for a model to have the flexibility to deal with changing, novel, and disappearing characteristics in the environment. The assumption that mixture parameters are static is a hinderance to the DP's use in such environments. Further, this assumption has the unfortunate effect that all the data must be processed in *batch* during inference, leading to ever-increasing computational costs as time goes on and more data is collected.

The dependent Dirichlet process (DDP) [48] is a BNP model that extends the DP and remedies the aforementioned problems. The fundamental notion of the DDP is that it is essentially a Markov chain of DPs, where the transitions between DPs allows mixture components to change, disappear, or to be newly created. This gives the DDP additional flexibility over the DP that is required in many autonomous planning system environments. Further, the Markov property of the system allows inference to consider data in a *batch-sequential* framework, where only small subsets of the total dataset ever need to be stored or processed at once.

There are a number of equivalent definitions of the DDP, but perhaps the most illustrative one (and the one upon which a large portion of the present work is built) is based on Poisson processes [12]:

Definition 2.2.3 *Let X be a measure space, and let $\psi : X \to [0, \infty)$ be an integrable intensity function on X, where $\Psi(B) = \int_B \psi$ is the mean measure for all measurable $B \subset X$. A* Poisson process $P \sim \Pi\mathrm{P}(\Psi)$ *on X is a random countable subset of X*

such that the number of points in $B \subset X$, $N(B) = |P \cap B|$ is a random number with a Poisson distribution $N(B) \sim \text{po}(\Psi(B))$.

Suppose that a Poisson process is defined over some space $X \times [0, \infty)$ (also known as a Gamma process or compound Poisson process). Then each atom i of the point process consists of a point $\theta_i \in X$ and a point $\pi_i' \in \mathbb{R}_+$ that may be thought of as a weight on θ_i. If the weights are then normalized, such that $\pi_i = \frac{\pi_i'}{\sum_i \pi_i'}$, the new point process G with the normalized weights is a Dirichlet process $G \sim \text{DP}(\alpha, H)$ with $\alpha = \Psi(X \times [0, \infty))$ and H is proportional to ψ with the part over the positive reals integrated out. This is referred to as a Poisson process construction of the Dirichlet process.

The reason why this construction is of particular importance is because a Poisson process can be probabilistically mutated in a number of ways such that the resulting point process is still a Poisson process. Thus, if a Dirichlet process is formulated in terms of its underlying Poisson process, such operations are performed on the underlying process, and then the weights are renormalized to be a Dirichlet process once again, one can think of the entire procedure simply as operations upon Dirichlet processes that preserve Dirichlet processes. The following operations preserve Poisson processes [90]:

Proposition 2.2.4 ***Superposition:*** *Let $P \sim \Pi\text{P}(\Psi)$, $Q \sim \Pi\text{P}(\Phi)$ be Poisson processes. Then $P \cup Q \sim \Pi\text{P}(\Psi + \Phi)$.*

Proposition 2.2.5 ***Subsampling:*** *Let $P \sim \Pi\text{P}(\Psi)$ be a Poisson process on X with mean measure Ψ and corresponding intensity ψ, let $q : X \to [0, 1]$, and let $b_i \sim \text{Be}(q(\theta_i))$ be a Bernoulli trial for every $\theta_i \in P$. Then $Q = \{\theta_i \in P : b_i = 1\}$ is a Poisson process, where $Q \sim \Pi\text{P}(\Phi)$ and $\Phi(B) = \int_B q\psi$.*

Proposition 2.2.6 ***Transition:*** *Let $P \sim \Pi\text{P}(\Psi)$ be a Poisson process on X with mean measure Ψ and corresponding intensity ψ. Then suppose $T : X \times X \to \mathbb{R}_+$, where $T(\cdot|\theta)$ is a probability distribution on X. For every $\theta_i \in P$, let $\theta_i' \sim T(\theta_i'|\theta_i)$. Then $Q = \{\theta_i'\} \sim \Pi\text{P}(\Phi)$ is a Poisson process with intensity $\phi = \int_X T(\theta'|\theta)\psi(\theta)$.*

When the basic operations on Poisson processes are ignored and the operations are formulated entirely on Dirichlet processes, superposition is denoted $\cup$, subsampling with the function q is denoted S_q, and transition with the function T is denoted T. The dependent Dirichlet process is then constructed as a Markov chain $D_t \sim \mathrm{DP}$ as follows:

$$\begin{aligned} D_0 &\sim \mathrm{DP}(\alpha_0, H_0) \\ D_{t+1} &= T\left(S_q\left(D_t\right)\right) \cup G_\nu \\ G_\nu &\sim \mathrm{DP}(\alpha_\nu, H_\nu) \end{aligned} \tag{2.11}$$

which has the aforementioned desired properties. Mixture components θ can be removed via S_q, move via T, and can be newly added via G_ν. Because at each time step t the model evolves using only operations that preserve the required properties of a Dirichlet process, it can be used as a prior over evolving mixture models. Furthermore, at each timestep, the illustration in Fig. 2-3 retains its accuracy in depicting the point process itself.

The posterior process of the dependent Dirichlet process given a collection of observations of the points is similar to (2.8), except that old mixture components from previous time steps need to be handled correctly. Given the observations of old transitioned points $\mathcal{T}_t$, new points $\mathcal{N}_t$, and points that were observed in past timesteps but not at timestep t, $\mathcal{O}_t$, the posterior is

$$G_t|\mathcal{T}_t, \mathcal{N}_t, \mathcal{O}_t \sim \mathrm{DP}\left(\nu_t + \sum_{\theta\in\mathcal{O}_t} q_{\theta t} c_{\theta t} T(\cdot|\theta) + \sum_{\theta\in\mathcal{T}_t} (c_{\theta t} + n_{\theta t})\delta_\theta + \sum_{\theta\in\mathcal{N}_t} n_{\theta t}\delta_\theta\right) \tag{2.12}$$

where ν_t is the measure for unobserved points at time step t, $c_{\theta t} = \sum_{\tau=1}^{t-1} n_{\theta t} \in \mathbb{N}$ is the number of observations of θ from previous time steps, $n_{\theta t} \in \mathbb{N}$ is the number of observations of θ at timestep t, and $q_{\theta t} \in (0, 1)$ is the subsampling weight on θ at timestep t. This posterior has a connection to the CRP that is similar to that between the DP and the CRP; both of which are exploited in the development of approximate inference techniques below.

2.2.3 Approximate BNP Inference

As mentioned earlier, conducting Bayesian inference amounts to solving (2.1) for the posterior distribution. On the surface this seems like a trivial exercise - however, even in simple cases, the denominator $p(y) = \int_\theta p(y|\theta)p(\theta)$ can be intractable to compute. Furthermore, $p(\theta)$ may not even be available in closed form. This is the case with many Bayesian nonparametric models; for example, in (2.8), there is no closed form for $p(G)$. Therefore, approximate inference techniques that circumvent these issues are required.

The most common inference procedures involving BNPs are Markov Chain Monte-Carlo (MCMC) methods, such as Gibbs sampling and the Metropolis-Hastings algorithm [16]. Although it can be hard to tell if or when such algorithms have converged, they are simple to implement, are guaranteed to provide samples from the exact posterior, and have been successfully used in a number of studies [14]. This is presently an area of active research, and other methods such as variational Bayesian inference [17] and particle learning [19] are currently under development. Due to the simplicity, popularity and generality of Gibbs sampling, this approximate inference technique will be discussed here; the reader is encouraged to consult the references for discussions of the other techniques.

The basic idea of Gibbs sampling is straightforward: given access to the conditional distributions of the latent parameters, sampling from their joint distribution may be approximated by iteratively sampling from each parameter's conditional distribution. For example, for a model with three parameters $\theta_1, \theta_2, \theta_3$ with the posterior $p(\theta_1, \theta_2, \theta_3|y)$ given the data y, posterior samples may be obtained by sampling $p(\theta_1|y, \theta_2, \theta_3)$, $p(\theta_2|y, \theta_1, \theta_3)$, and $p(\theta_3|y, \theta_1, \theta_2)$ iteratively. By doing this, a Markov chain whose equilibrium probability distribution is that of the full joint probability distribution is created. Practically speaking, one must first let the Markov chain reach its equilibrium distribution by sampling from the conditional distributions T_1 times without recording any of the samples, and then take a sample every T_2 iterations from the algorithm to be a sample from the joint distribution. Then, given the

approximate joint distribution samples, one can estimate whatever statistics are of interest, such as the mean or variance of the distribution.

In the world of Bayesian nonparametrics, Gibbs sampling solves the issue of not having a closed-form prior $p(G)$. For mixture models, one expresses the latent variable G in terms of the parameters θ_k and data label assignments z_i, and iteratively samples the conditional distributions $p(\theta_k|\theta_{-k}, z, y)$ and $p(z_i|\theta, z_{-i}, y)$ to get samples from $p(\theta, z|y)$.

For the Dirichlet process mixture model (shown with a normal likelihood), the procedure is quite simple, as the CRP provides us with the required probability distributions:

$$\begin{aligned} p(z_i = k|y, z_{-i}, \theta) &\propto \begin{cases} \frac{n_k}{N+\alpha-1}\mathcal{N}(y_i;\theta_k) & k \leq K \\ \frac{\alpha}{N+\alpha-1}\int \mathcal{N}(y_i;\theta_k)H(\theta_k)\mathrm{d}\theta_k & k = K+1 \end{cases} \\ p(\theta_k|y, z, \theta_{-k}) &\propto \prod_{i|z_i=k} \left[\mathcal{N}(y_i|\theta_k)\right] H(\theta_k) \end{aligned} \tag{2.13}$$

where H is the prior over the parameters. Setting H to a normal distribution yields a conjugate prior, and the integral has a closed-form solution. In the case of non-conjugate priors, the Metropolis-Hastings [16] algorithm, a generalization of Gibbs sampling, may be used.

Gibbs sampling for the dependent Dirichlet process mixture is slightly more complex. Once again, the CRP provides the required distributions; however, one must keep track of which mixture components fall into a number of bins:

- New components ($\mathcal{N}_t$) are ones which were sampled from G_ν, the innovation process. There must be at least one datapoint assigned to the cluster.
- Old, instantiated components ($\mathcal{T}_t$) are ones which were previously observed in an earlier time step, and now have at least one datapoint assigned to them in the current time step. This means they must have survived subsampling via S_q, and were transitioned via T for however long they were not observed.
- Old, uninstantiated components ($\mathcal{O}_t$) are ones which were previously observed

in an earlier time step, and have no observations assigned to them in the current time step. It is uncertain whether they were subsampled out via S_q, or whether there are just no datapoints that are assigned to them.

Taking these different types of label assignment into account, the two steps of Gibbs sampling for the dependent Dirichlet process are:

$$p(z_{it} = k|\dots) \propto \begin{cases} \alpha_t \int \mathcal{N}(y_{it}, \theta_{kt}) H(\theta_{kt}) \mathrm{d}\theta_{kt} & k = K+1 \\ (c_{kt} + n_{kt}) \mathcal{N}(y_{it}, \theta_{kt}) & n_{kt} > 0 \\ q_{kt} c_{kt} \mathcal{N}(y_{it}, \theta_{k(t-\Delta t_k)}) & n_{kt} = 0 \end{cases} \tag{2.14}$$

$$\mathrm{p}(\theta_{kt}|y_t, z_t) \propto \prod_{i:z_{it}=k} \left[\mathcal{N}(y_{it}|\theta_{kt})\right] p(\theta_{kt}) \tag{2.15}$$

where

$$p(\theta_{kt}) \propto \int_{\theta_{k(t-\Delta t_k)}} T(\theta_{kt}|\theta_{k(t-\Delta t_k)}) p(\theta_{k(t-\Delta t_k)}) \mathrm{d}\theta_{k(t-\Delta t_k)} \tag{2.16}$$

and where $p(\theta_{k(t-\Delta t_k)})$ is incrementally updated by keeping track of the sufficient statistics of each cluster created by the algorithm.

2.3 Markov Decision Processes

Markov decision processes (MDPs) are a general framework for solving sequential decision-making problems, and are capable of representing many different planning problems involving uncertainty. They find application in a wide variety of fields, ranging from robotics and multiagent systems[91] to search, queueing, and finance[20]. A thorough description of MDPs and related algorithms may be found in [92].

2.3.1 Markov Chains

In order to discuss the MDP framework and their properties that are useful in the context of this thesis, Markov chains must first be mentioned:

Definition 2.3.1 *A Markov chain is a tuple $\langle \mathcal{S}, P \rangle$ where $\mathcal{S}$ is a set of states, and $P : \mathcal{S} \times \mathcal{S} \to [0, 1]$ is a transition model defining the probability $P(s'|s)$ of transitioning from a state $s \in \mathcal{S}$ to another state $s' \in \mathcal{S}$ in a single step, conditionally independent of all states previous to the chain being in state s.*

When a Markov chain is realized, it generates a sequence of states $s_1, s_2, \ldots$ that follow the transition probabilities in P. Due to the definition of Markov chains, however, the joint probability of a sequence can be decomposed in a particular way:

$$p(s_1, s_2, \ldots, s_n) = p(s_1) \prod_{i=2}^{n} p(s_i|s_{i-1}) \tag{2.17}$$

where $p(s_1)$ is the probability distribution over the starting state s_1. This decomposition is often helpful in avoiding exponential complexity growth when marginalizing over the sequence of states in the realization of a Markov chain.

Since P is parameterized by two states, it can be written as a matrix $\mathbf{P}$, where $\mathbf{P}_{ij}$ is the probability from transitioning from state i to state j. This is useful when propagating a state distribution through a Markov chain for a certain number of steps:

$$\mathbf{s}_t = (\mathbf{P}^T)^t \mathbf{s}_0 \tag{2.18}$$

where $\mathbf{s}_t$ is the distribution over the state of the Markov chain at time step t.

There are many classifications and properties of Markov chains; one such kind that is of particular importance to this thesis is defined as follows:

Definition 2.3.2 *An absorbing Markov chain is one in which there exists a set of states $\mathcal{S}_{abs} \subset \mathcal{S}$ such that when the Markov chain enters any state $s \in \mathcal{S}_{abs}$, it remains in s for all future time steps, and it is possible to go from any state $s' \notin \mathcal{S}_{abs}$ to a state $s \in \mathcal{S}_{abs}$ in a finite number of steps.*

In absorbing Markov chains, the matrix $\mathbf{P}$ may be written as follows:

$$\mathbf{P} = \begin{bmatrix} \mathbf{Q} & \mathbf{R} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \tag{2.19}$$

where $\mathbf{I}$ is the identity matrix for all the rows corresponding to states $s \in \mathcal{S}_{\text{abs}}$, and $\mathbf{0}$ is the zero matrix.

2.3.2 Markov Decision Processes (MDPs)

Markov decision processes (MDPs) extend Markov chains by adding in the concept of *actions* and *rewards*. In each state, a decision-maker can select which action to take in order to maximize some quantity related to the overall future reward gathered by the system. Defining this precisely, a Markov decision process is a tuple $\langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$ where: $\mathcal{S}$ is a set of states; $\mathcal{A}$ is a set of actions; $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is a transition model defining the probability of entering state $s' \in \mathcal{S}$ given a previous state $s \in \mathcal{S}$ and an action $a \in \mathcal{A}$; $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is a reward function; and $\gamma \in (0, 1)$ is a discount factor for the rewards.

A deterministic policy for an MDP is a mapping $\pi : \mathcal{S} \to \mathcal{A}$. Any policy induces a value function V^π on the states of the MDP,

$$V^\pi(s_0) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(s_t), s_{t+1}) \middle| \pi, s_0\right], \tag{2.20}$$

which is equal to the expectation of the γ-discounted reward received when executing the policy π over an infinite horizon, starting from state s_0. Note that given a fixed policy, an MDP is reduced to a Markov reward process (MRP), a Markov chain with a stochastic state-dependent reward received at each time step.

Maximizing the value function of an MDP with respect to the actions taken is the fundamental problem of planning in the MDP framework. Many algorithms exist to solve this problem, such as exact methods (e.g. value iteration and policy iteration[92]), approximate dynamic programming (e.g. UCT [93]), and reinforcement learning (e.g. Dyna [73]). While MDP solvers generally have worst-case polynomial time solutions in the number of states and actions, most methods suffer the "curse of dimensionality" [92]: when used for systems with high dimension (such as multiagent systems), the state space grows in size exponentially with the number of dimensions, and thus the worst-case solution time grows exponentially as well. This issue forms

one of the core motivations of this thesis.

2.3.3 Linear Function Approximation

Many planning problems formulated as MDPs have very large state spaces $\mathcal{S}$, and it is impractical to use an exact description of $\mathcal{S}$ (e.g. a tabular representation) due to memory and computational limitations. For example, a very simple multiagent system with N_a agents, where each agent exists on a 10×10 grid, has 10^{2N_a} states; for $N_a \approx 4$ this system becomes too large for a tabular representation on currently available hardware. Instead, for large state spaces, a function ϕ is used to map the state space into a lower-dimensional *feature space*, and then one performs operations in the new smaller space. Linear function approximation is a common approach to take [73, 94–96], where $\phi : \mathcal{S} \to \mathbb{R}^m$ maps each state to m feature values. Using this mapping, any scalar field $f : \mathcal{S} \to \mathbb{R}$ (e.g. the reward function R) can approximated by Φf_Φ in the feature space, where $f_\Phi \in \mathbb{R}^m$:

$$\Phi = \begin{bmatrix} \phi(s_1) & \cdots & \phi(s_{|\mathcal{S}|}) \end{bmatrix}^T, \Phi f_\Phi \approx f = \begin{bmatrix} f(s_1) & \cdots & f(s_{|\mathcal{S}|}) \end{bmatrix}^T. \tag{2.21}$$

2.3.4 Model Learning

The *model* of an MDP is defined as its dynamics P and its reward R. In many planning scenarios, these quantities are not well-known a priori; for example, consider a glider motion planning problem where the reward R in each state is the observed altitude increase. A pilot may not know the locations of thermal updrafts a priori (i.e. R is not well-known), and must explore the airspace to find them in order to stay aloft longer. This paradigm of observing samples from an MDP, using them to update the model, and planning based on the updated model is called model-based reinforcement learning [73]. In this thesis, model learning only occurs given a fixed policy; consequently, $\mathcal{A}$ is removed from consideration (yielding an MRP instead of an MDP). This situation occurs as a subcomponent of many approximate planning algorithms, such as the policy evaluation step of policy iteration [73, 97].

Further, model learning in this thesis is restricted to learning an unknown scalar field $f : \mathcal{S} \to \mathbb{R}$ (e.g. the reward model R, or perhaps a state-dependent uncertain parameter in P [97]) for simplicity of exposition. However, it is straightforward to extend this work to the case of learning P and R, including the variation over the actions $\mathcal{A}$, without any partial knowledge.

Given the linear feature matrix Φ, the approximation $f \approx \Phi f_\Phi$ can be found by minimizing the squared norm between f and Φf_Φ, weighted by the stationary probability of each state:

$$f_\Phi = \underset{w}{\operatorname{argmin}} \|\Sigma^{\frac{1}{2}}(f - \Phi w)\|_2^2 \implies f_\Phi = \left(\Phi^T \Sigma \Phi\right)^{-1} \Phi^T \Sigma f \tag{2.22}$$

where Σ is the diagonal matrix representation of the stationary distribution on $\mathcal{S}$. Since f is not known, the model learning problem is to deduce f_Φ given a set of observation episodes from the MRP. Let the set of observed episodes be $\{y_i\}_{i=1}^N$, where $y_i = \{(s_{i1}, f_{i1}), \ldots, (s_{iT_i}, f_{iT_i})\}$, f_{ij} is a noisy unbiased estimate of $f(s_{ij})$, and $\phi(s_{ij}) \equiv \phi_{ij}$ for brevity. Then the problem of minimizing the sum of squared differences $|f_{ij} - {f_\Phi}^T \phi_{ij}|^2$ over all the observed samples is solved via

$$f_\Phi = \left[\sum_{i,j} \phi_{ij} \phi_{ij}^T\right]^{-1} \left[\sum_{i,j} \phi_{ij} f_{ij}\right], \tag{2.23}$$

where the sums are over the range $j \in \{1, \ldots, T_i\} \, \forall \, i \in \{1, \ldots, N\}$. As N increases, the solution to (2.23) approaches the solution to (2.22). To replace f with learning P and R (as mentioned earlier), simply replace f with R directly (as they are both $|\mathcal{S}|$-dimensional vectors), and add P by considering feature prediction errors and swapping vector norms for Frobenius norms where required. For a more thorough discussion of single model-based RL with a linear representation, the reader is directed to the analysis by Parr et al. [94].

2.3.5 Adaptive Linear Representation

The ϕ function defines the subspace in which f_Φ is represented. It is often difficult to know a priori what choice of ϕ yields a compact representation of $\mathcal{S}$, while still preserving its major structure. Thus, methods that adapt ϕ based on samples observed from an MDP/MRP are useful when performing model learning, as they provide an automated tradeoff between representational quality and computational expense. There are a number of different representation expansion algorithms; this thesis primarily uses the batch incremental Feature Dependency Discovery algorithm (Batch-iFDD) [72]. Given a binary representation $\Phi \in \{0,1\}^{|\mathcal{S}|\times m}$, each column of Φ corresponds to the value of specific feature for all states. During each iteration, Batch-iFDD inspects the binary conjunction of all pairs of columns of Φ and appends a conjunction θ^* to Φ according to

$$\theta^* = \underset{\theta \in \texttt{colpair}(\Phi)}{\text{argmax}} \frac{|\sum_{i,j:\theta_{ij}=1} \epsilon_{ij}|}{\sqrt{\sum_{i,j:\theta_{ij}=1} 1}}, \tag{2.24}$$

in which $\texttt{colpair}(\Phi)$ is the set of conjunctions of pairs of columns of Φ, θ_{ij} is the conjunction feature value for state s_{ij}, and ϵ_{ij} is some predictive error related to a quantity of interest. In the original work, Batch-iFDD was used to approximate the value function, hence ϵ_{ij} was the temporal differences on step j of episode i [72]. This thesis utilizes Batch-iFDD to expand a representation used to capture an MDP model, and thus ϵ_{ij} is the prediction difference $\epsilon_{ij} = f_{ij} - {f_\Phi}^T \phi_{ij}$.

2.4 Multiagent Task Allocation

Multiagent task allocation (or assignment) problems generally involve a group of agents coordinating to complete a set of tasks, such that there are no conflicts in the assignments of tasks to agents. It is typically beneficial for agents not to plan myopically by selecting one task at a time, and instead to take on more than one task in an assignment. Therefore, the allocation problem considers maximizing performance given sequences of tasks for each agent, referred to as their *path* or *task sequence.*

Multiagent task allocation problems tend not to pose the curse of dimensionality issues that Markov decision processes do; this is because task allocation problems are typically formulated at a much higher level, focusing on coordination between agents rather than focusing on selecting individual, detailed actions.

In this work, multiagent task allocation is formulated as a mathematical program:

$$\begin{aligned} \max_{\mathbf{x},\boldsymbol{\tau}} \quad & \sum_{i=1}^{N_a}\sum_{j=1}^{N_t} F_{ij}(\mathbf{p}_i,\boldsymbol{\tau})\; x_{ij} \\ \text{s.t.} \quad & \mathbf{H}(\mathbf{x},\boldsymbol{\tau}) \leq \mathbf{d} \\ & \mathbf{x} \in \{0,1\}^{N_a \times N_t},\; \boldsymbol{\tau} \in \{\mathbb{R}^+ \cup \emptyset\}^{N_a \times N_t} \end{aligned} \tag{2.25}$$

where $\mathbf{x}$ is a set of $N_a \times N_t$ binary decision variables, x_{ij}, which are used to indicate whether or not task j is assigned to agent i; $\mathbf{p}_i$ is the path which represents the order in which agent i services its assigned tasks; $\boldsymbol{\tau}$ is the set of variables τ_{ij} indicating when agent i will service its assigned task j (where $\tau_{ij} = \emptyset$ if task j is not assigned to agent i); F_{ij} is the score function for agent i servicing task j given the path and service times; and $\mathbf{H} = [h_1, \ldots, h_{N_c}]^T$, with $\mathbf{d} = [d_1, \ldots, d_{N_c}]^T$, define a set of N_c possibly nonlinear constraints of the form $h_k(\mathbf{x}, \boldsymbol{\tau}) \leq d_k$ that capture transition dynamics, resource limitations, etc. This generalized problem formulation can accommodate several different design objectives and constraints commonly used in multiagent decision making problems. One could of course solve this mixed integer nonlinear program generally with a combinatorial optimization algorithm. An important observation is that, in (2.25), the scoring and constraint functions explicitly depend on an agent's ordered sequence of task assignments, as well as those task service times ($\mathbf{p}_i$ and $\boldsymbol{\tau}$, respectively). This property makes the general mixed-integer programming problem very difficult to solve (NP-hard) exactly due to the inherent system interdependencies [98].

This motivates approximate solution techniques for the multiagent task allocation problem. One such technique that is of particular relevance to this thesis is the *sequential greedy* allocation algorithm, shown in Alg. 1. This algorithm works by building up the agents' task sequences one task at a time. In each iteration, it inserts

Algorithm 1 Sequential Greedy

Input: Set of tasks $\mathcal{T}$, agents $\mathcal{Z}$

$\forall i \in \mathcal{Z} : \mathbf{p}_i \leftarrow \{\}, J_{i,\text{prev}} \leftarrow 0$
while $\mathcal{T} \neq \emptyset$ **do**
  for $i \in \mathcal{Z}, j \in \mathcal{T}, k \in \mathbf{p}_i$ **do**
    $J_{ijk} \leftarrow$EVALTASKSEQ$(\mathbf{p}_i \oplus_k j)$
  end for
  $\hat{i}, \hat{j}, \hat{k} \leftarrow \arg\max J_{ijk} - J_{i,\text{prev}}$
  $\mathcal{T} \leftarrow \mathcal{T} \backslash \hat{j}$, $J_{\hat{i},\text{prev}} \leftarrow J_{\hat{i}\hat{j}\hat{k}}$, $\mathbf{p}_{\hat{i}} \leftarrow \mathbf{p}_{\hat{i}} \oplus_{\hat{k}} \hat{j}$
end while
return $\mathbf{p}_i \, \forall i \in \mathcal{Z}$

every remaining unassigned task j into every possible slot in every agent's current task sequence (insertion at index k is denoted $\oplus_k$). Testing each insertion is done via the function EVALTASKSEQ, which evaluates the task sequence. Finally, the best task/agent/index combination is selected based on the marginal gain in score, and the iteration loops until all tasks have been assigned. It is easy to show that this algorithm is polynomial in the number of agents and tasks, and decentralized formulations of this algorithm have been created that mimic its greedy solutions and provide performance guarantees under a wide variety of conditions [24]. Furthermore, while the original version of these algorithms have assumed deterministic task sequence evaluation schemes, they have been extended to generate plans that are robust to various forms of uncertainty [22, 29, 99].

Chapter 3

Multiagent Allocation of Markov Decision Process Tasks

3.1 Overview

Producing task assignments for multiagent teams often leads to an exponential growth in the decision space as the number of agents and objectives increases. One approach to finding a task assignment is to model the agents and the environment as a single Markov decision process, and solve the planning problem using standard MDP techniques. However, both exact and approximate MDP solvers in this environment struggle to produce assignments even for problems involving few agents and objectives. Conversely, problem formulations based upon mathematical programming typically scale well with the problem size at the expense of requiring comparatively simple agent and task models. This chapter presents a combination of these two formulations by modeling task and agent dynamics using MDPs, and then using optimization techniques to solve the combinatorial problem of assigning tasks to agents. The computational complexity of the resulting algorithm is polynomial in the number of tasks and is constant in the number of agents. Simulation results are provided which highlight the performance of the algorithm in a grid world mobile target surveillance scenario, while demonstrating that these techniques can be extended to even larger tasking domains.

This chapter is based on the paper "Multiagent Allocation of Markov Decision Process Tasks" [100], which was presented at the 2013 American Controls Conference in collaboration with L. Johnson and J. How.

3.2 Introduction

The goal of autonomous planning for teams of agents is to achieve mission objectives while satisfying all problem-specific constraints. Task assignment algorithms for multiagent teams thus require two components: 1) a model that describes how agents and tasks interact in a possibly stochastic world, and 2) an objective function that specifies which assignment combinations are most beneficial to the fleet and guides the task allocation process.

Prior work of particular interest to solving this problem exists in both the fields of mathematical programming-based task allocation algorithms and Markov decision process (MDP) planning algorithms. Some mathematical programming-based task allocation formulations are designed to provide assignments when task models have a constant, deterministic state (e.g. visiting a set of waypoints) [27, 28]. Recent approaches have solved a similar problem for distributed multiagent multiassignment environments efficiently by using sequential greedy algorithms [25]. These greedy assignment algorithms have also been extended to include the effects of stochastic parameters on the overall task assignment process [29]. The approach in [29] relies on sampling a set of particles from the distributions of stochastic parameters and making informed decisions based on those particles. Thus, while scalable coordination for multiagent teams with stochastic parameters has been addressed within this framework, more formal problem statements involving generalized stochastic interactions between agents and tasks have not.

In the framework of Markov decision processes, explicit and guaranteed multiagent coordination for general problems is well known to exhibit exponential computational scaling with respect to the number of agents in the team [21]. On the other hand, a number of previous studies have explored the benefits of decomposing MDPs into

smaller *weakly coupled* [32] concurrent processes, solving each individually, and merging the solutions to form a solution of the larger problem [33, 34]. Building on that work, several researchers have considered problems where shared resources must be allocated amongst the concurrent MDPs [32, 35]; generally, in such problems, the allocation of resources amongst the MDPs determines which actions are available in each MDP, or the transition probabilities in each MDP. Some approaches involve solving resource allocation or coordination problems by modelling them within the MDP itself [36, 37]. Assignment based decompositions have also been studied [38], where cooperation between agents is achieved via coordinated reinforcement learning [39]. However, none of these approaches have considered the impact of previous tasks on future tasks. This is of importance in multiagent multi-assignment planning problems; for instance, if an agent is assigned a task involving a high amount of uncertainty, that uncertainty should be properly accounted for in all of its subsequent tasks (whose starting conditions depend on the final state of the uncertain task). This allows the team to hedge against the risk associated with the uncertain task, and allocate the subsequent tasks accordingly. Past work has focused primarily on parallel, infinite-horizon tasks [32, 35], and have not encountered the cascading uncertainty that is characteristic of sequential bundles of uncertain tasks.

An insight can be made by exploiting the natural structure of task allocation problem formulations. Often in these environments, given a conflict-free assignment of tasks to agents, agents engaged in different tasks operate independently; similarly, subsets of tasks often evolve independently from one another. Approximate mathematical programming-based solvers are adept at utilizing such independence to plan efficiently and tractably in scenarios involving large numbers of agents and tasks, while MDPs are able to explicitly formulate task and agent models with uncertainty in a precise and general form. This chapter combines the strengths of these two approaches and considers the problem of multiagent distributed task allocation with generalized stochastic task models, where each agent may be assigned a list of sequential tasks.

3.3 Problem Statement

An important property exploited in this work is that there are few restrictions on the function F_{ij} in the task allocation problem defined in (2.25). In particular, evaluating F_{ij} may involve the manipulation of probability distributions, sampling from them, or the computation of their statistics. If a distribution over F_{ij} itself is provided or can be easily computed given $\boldsymbol{p}_i$ and $\boldsymbol{\tau}$, uncertainty may be accounted for in the allocation scheme [22]. However, computing such a distribution is often difficult and relies on problem-specific algorithms.

Markov decision processes, on the other hand, provide an evaluation scheme for complex tasks with uncertain outcomes that can be generalized across a wide variety of scenarios. An insight that is useful here is that *each individual task* can be formulated as an MDP, rather than formulating the entire problem as a single MDP. By doing so, a valid score function F_{ij} can be constructed for each agent i and task j pair using the discounted sum of rewards in the corresponding MDP (given the starting conditions of the task). Thus, the task allocation optimization problem may be rewritten as follows:

$$\max_{\mathbf{x},\mathbf{p}} \quad \sum_{i=1}^{N_a}\sum_{j=1}^{N_t} \mathbb{E}_{s_{ij},\tau_{ij}}[V_{ij}^{\pi}(s_{ij},\tau_{ij})|\mathbf{p}]\; x_{ij} \qquad (3.1)$$
$$\mathbf{x} \in \{0,1\}^{N_a \times N_t}$$

where s_{ij} and τ_{ij} are random variables defining the starting state and time of agent i servicing task j. V_{ij}^{π} is the value function for the MDP in which agent i services task j, which provides a valid score function for a realized state and time pair. This mathematical program makes the complexity of the problem formulation explicit. A solution algorithm requires a method to compute the value functions for all task and agent pairs, a method to compute the distribution over each s_{ij}, τ_{ij} pair, and a combinatorial optimization algorithm to maximize the multiagent team performance.

Not all MDP formulations are inherently adapted for use in this task allocation framework, and there are issues with their direct inclusion. While the value function

V^π is a natural scoring mechanism for infinite-horizon MDPs, tasks will have a finite, stochastic duration. Furthermore, the uncertain outcomes of earlier tasks affect the performance of the team on later tasks. These issues will be addressed in the following sections.

3.4 MDP Task Model

The general definition of MDPs has insufficient structure to use them as a model for individual tasks. Thus, in order to use task allocation algorithms with an MDP-based task model, a more restricted class of MDPs will be defined, called *Markov decision process tasks*. This formulation captures the important structure of tasks in task assignment problems, such that MDPs may be utilized to efficiently compute allocations.

An MDP task for task j and agent i is a tuple $\langle \mathcal{D}_j, \mathcal{G}_i, \mathcal{A}_i, P_{ij}, Q_{ij}, R_{ij}, \gamma \rangle$ where:

- $\mathcal{D}_j$ is a set of *task states for task* j,
- $\mathcal{G}_i$ is a set of *agent states* for agent i, (let $\mathcal{S}_{ij} = \mathcal{G}_i \times \mathcal{D}_j$),
- $\mathcal{A}_i$ is a set of actions that agent i may take,
- $P_{ij} : \mathcal{S}_{ij} \times \mathcal{A}_i \times \mathcal{S}_{ij} \to [0, 1]$ is a transition model $P_{ij}(s'|a, s)$ over $s' \in \mathcal{S}_{ij}$ given $s \in \mathcal{S}_{ij}$ and $a \in \mathcal{A}_i$ for when the task is being executed by the agent,
- $Q_{ij} : \mathcal{S}_{ij} \times \mathcal{D}_j \to [0, 1]$ is a transition model over $d' \in \mathcal{D}_j$ given $s \in \mathcal{S}_{ij}$ for before the execution occurs,
- $R_{ij} : \mathcal{S}_{ij} \times \mathcal{A}_{ij} \times \mathcal{S}_{ij} \to \mathbb{R}$ is a reward function,
- $\gamma \in (0, 1)$ is a reward discount factor,
- $\mathcal{D}_{j,\text{abs}} \subset \mathcal{D}_j$ is a nonempty set of reachable, absorbing *terminal states* (let $\mathcal{S}_{ij,\text{abs}} = \mathcal{G}_i \times \mathcal{D}_{j,\text{abs}}$),
- and if the optimal policy $\pi_{ij} : \mathcal{S}_{ij} \to \mathcal{A}_i$ for the MDP task is executed, the expected time to reach a terminal state is finite (i.e. tasks must end)

MDP tasks are set apart from typical MDPs by the requirement that the state space is decomposed into agent and task states, the absorbing nature of tasks, and the prior-to-execution dynamics Q_{ij}. With respect to the absorbing condition, if the combined state of an agent and task ever enters $s \in \mathcal{S}_{ij,\text{abs}}$, it will (conceptually) remain in $\mathcal{S}_{ij,\text{abs}}$ for all future timesteps, and thus the task is said to be completed. Furthermore, the requirement that an optimal policy results in an absorbing Markov chain means that the expected completion time for each task is finite [101]. These two requirements together enforce that it must be possible to complete each task, and that an agent executing the task optimally is expected to complete the task in finite time. The prior-to-execution dynamics Q_{ij} are required because the task state might depend on the agent state prior to execution of that task, and the behavior prior to execution might differ from the behavior during execution. It is not used in the computation of a policy for agent i executing task j, just in the during the evaluation of the score of task j.

3.5 Task Sequence Evaluation

In this section, the evaluation of the objective in (3.1) given a candidate $\mathbf{x}$ and $\mathbf{p}$ is considered. In this work we make three important assumptions about how the agents and tasks interact:

1. The rewards received by different agents are independent given a task assignment $\mathbf{x}$. This means that the scores each agent receives for completing tasks are not dependent on the behavior of any other agent, and allows each agent to evaluate its task sequence without requiring information from other agents.

2. Policies $\boldsymbol{\pi} = \{\pi_{ij}\}$ are provided for all combinations of agent i and task j before the assignment algorithm starts. This is not unreasonable, because the size of a single combined task-agent state space is likely small and does not depend on the number of tasks or agents in the assignment problem, only the number of unique task and agent *types*. Even if the agents are indeed heterogeneous, obtaining a

policy for all task and agent combinations has polynomial complexity $O(\bar{N}_a \bar{N}_t)$ in the number of unique task types $\bar{N}_t$ and agent types $\bar{N}_a$.

3. Agents focus on one task at a time. During an individual task execution, the policy executed does not take into account the objectives of any future tasks.

With these assumptions, the goal is to compute the value of a sequence of tasks for a single agent, where that later tasks depend on the outcomes of previous tasks. Given that this section considers the sequence of a single agent, agent indices i are dropped from quantities, and the task indices j are replaced with their positions k in the sequence $\mathbf{p}_i$. Let $s_k = s_{ij_k} \in \mathcal{S}_{ij_k}$ be the starting state for agent i servicing task j_k, where task j_k is the k^{th} task in its sequence. Let $\tau_k = \tau_{ij_k} \in \mathbb{Z}$ be the timestep at which that agent begins to service that task. Let the sequence of tasks be of length n, i.e. $k \in \{0, \ldots, n-1\}$. Finally, let r_t be the reward received by agent i at timestep t. Referring to the optimization problem statement (3.1), the proposed evaluation function for a sequence of tasks for a given agent is the sum of the expected values of its tasks. If this evaluation function is to be used, this should correspond to the value of the overall Markov process describing the entire sequence of tasks. Indeed it does; consider the expected sum of discounted rewards over a single agent's task sequence,

$$\mathbb{E}\left[\sum_{t=\tau_0}^{\infty} \gamma^t r_t \middle| s_0, \tau_0\right] = \mathbb{E}\left[\sum_{k=0}^{n-1} \sum_{t=\tau_k}^{\tau_{k+1}-1} \gamma^t r_t \middle| s_0, \tau_0\right]. \tag{3.2}$$

Equation (3.2) makes the breakdown of the rewards received into each individual task explicit; the k^{th} task in the sequence is responsible for the rewards received in the time interval $[\tau_k, \tau_{k+1} - 1]$. While s_0 and τ_0 are known precisely, s_k and τ_k for $k \geq 1$ in general are stochastic due to the uncertainty in the outcomes of previous tasks. This stochasticity may be made clear by rearranging (3.2) as follows:

$$\mathbb{E}\left[\sum_{k=0}^{n-1} \sum_{t=\tau_k}^{\tau_{k+1}-1} \gamma^t r_t \middle| s_0, \tau_0\right] = \sum_{k=0}^{n-1} \mathbb{E}\left[\sum_{t=\tau_k}^{\tau_{k+1}-1} \gamma^t r_t \middle| s_0, \tau_0\right] \tag{3.3}$$

$$= \sum_{k=0}^{n-1} \sum_{s_k, \tau_k} \mathrm{p}(\tau_k, s_k | \tau_0, s_0) \mathbb{E}\left[\sum_{t=\tau_k}^{\tau_{k+1}-1} \gamma^t r_t \middle| s_k, \tau_k\right] \tag{3.4}$$

$$= \sum_{k=0}^{n-1} \sum_{s_k, \tau_k} \mathrm{p}(\tau_k, s_k | \tau_0, s_0) \gamma^{\tau_k} V_{ij}^{\pi}(s_k) \tag{3.5}$$

where the value function for the policy of agent i executing task j starting at time 0 is $V_{ij}^{\pi}(s_k) : \mathcal{S}_{ij} \rightarrow \mathbb{R}$. It is noted that (3.5) follows from (3.4) because the k^{th} task enters an absorbing state once it terminates, and (conceptually) provides 0 reward for all future time; thus, the sum over $[\tau_k, \tau_{k+1} - 1]$ is replaced by a sum over $[\tau_k, \infty]$ and the definition of the value function from (2.20) may be used with $t_0 = 0$. Therefore, the objective in (3.1) equals the expected value of the discounted sum of rewards, and the value of a sequence of tasks may be constructed from the value functions for the individual tasks given stationary policies for those tasks.

3.6 Computing Starting State/Time Distributions

In general task allocation scenarios, the computation of $\mathrm{p}(s_k, \tau_k | s_0, \tau_0)$ is intractable due to the exponentially increasing number of outcomes of previous tasks. However, because of the Markov property of the framework presented herein, this distribution can be computed efficiently by propagating distributions forward in time one step at a time, using low-cost iterative updates.

Let $s_k = (g_k, d_k)$, where g_k, d_k are the agent, task component of s_k, respectively. Assume that $\mathrm{p}(s_k, \tau_k)$ (with conditional quantities implicit for brevity) has been computed, and that $\mathrm{p}(s_{k+1}, \tau_{k+1})$ must now be computed. There are two types of propagation udpates that are required given $p(s_k, \tau_k)$: 1) An update that determines the distribution over agent starting state g_{k+1} and time τ_{k+1} for the $k+1^{\text{th}}$ task by using the policy for the k^{th} task; and 2) An update that determines the distribution over d_{k+1} so that it may be combined with g_{k+1} to form s_{k+1}, and complete the iteration. These two updates will be referred to as the "on-execution update" and "prior-to-execution update", respectively.

3.6.1 The On-Execution Update

Suppose the distribution $\mathrm{p}(s_k, \tau_k)$ is known, and stored as a discrete distribution $\mathrm{p}(\tau_k)$ and a distribution $\mathrm{p}(s_k|\tau_k)$ (denoted $\mathbf{s}_{\tau_k}$ if expressed as a vector) for each element in the support of $\mathrm{p}(\tau_k)$. Since $\mathrm{p}(\tau_k)$ has theoretically infinite support, suppose it is truncated to the K_τ values with the highest probability. The following section will show how to compute $\mathrm{p}(g_{k+1}|\tau_{k+1})$ (denoted $\mathbf{g}_{\tau_{k+1}}$ if expressed as a vector) and $p(\tau_{k+1})$.

Define the state distribution:

$$\mathbf{s}_{\tau_k}[t] = p(s|t = t' + \tau_k, \mathbf{s}_{\tau_k}) = (\mathbf{P}_k^T)^{t'}\mathbf{s}_{\tau_k} \tag{3.6}$$

where $t \in \mathbb{N}$ and $\mathbf{P}_k$ is the Markov transition matrix corresponding to the k^{th} MDP task combined with its policy. This gives a state distribution parametrized by time for every value in the support of τ_k. For a feasible computation the domain of $\mathbf{s}_{\tau_k}[t]$ must be truncated from $t \in \mathbb{N}$. Let $\nu \in (0, 1)$ be a threshold probability, and compute $\mathbf{s}_{\tau_k}[t]$ for increasing t until the probability of having completed the k^{th} task is at least ν. In other words, the iteration in (3.6) is computed until at iteration N

$$\sum_{t=0}^{N} \mathrm{p}(\tau_{k+1} = t + \tau_k|\tau_k) > \nu. \tag{3.7}$$

Next define a task completion probability vector for task k as $\mathbf{C}_k = \mathbf{I}_{nonabs}\mathbf{P}_k\mathbf{1}_{abs}$, where $\mathbf{I}_{\text{nonabs}}$ is the identity matrix modified to have diagonal 0s in all rows corresponding to absorbing states, and $\mathbf{1}_{\text{abs}}$ is a column of 1s modified to have 0s in all rows corresponding to nonabsorbing states. The indicies of this vector correspond to the states in s_k, and each element represents the probability of transitioning from a non-absorbing state to an absorbing state in one time step. $\mathbf{C}_k$ takes the value of 0 for all absorbing states. The probability distribution for τ_{k+1} given τ_k can then be written as:

$$\mathrm{p}(\tau_{k+1} = t + \tau_k|\tau_k) = \mathbf{C}_k^T\mathbf{s}_{\tau_k}[t-1], t \in \mathbb{N} \tag{3.8}$$

τ_k can then simply be marginalized out to create the distribution:

$$\mathrm{p}(\tau_{k+1}) = \sum_{\tau_k} \mathrm{p}(\tau_k)\mathbf{C}^T \mathbf{s}_{\tau_k}[\tau_{k+1} - \tau_k - 1]. \tag{3.9}$$

Similarly, the agent state distribution at task completion can be computed using

$$\mathbf{g}_{\tau_{k+1}} = \sum_d \sum_{\tau_k} \frac{\mathbf{C} \circ s_{\tau_k}[\tau_{k+1} - \tau_k - 1]}{\mathbf{1}^T \mathbf{C} \circ s_{\tau_k}[\tau_{k+1} - \tau_k - 1]} \mathrm{p}(\tau_k | \tau_{k+1}) \tag{3.10}$$

where $\circ$ refers to element wise multiplication, $\mathrm{p}(\tau_k|\tau_{k+1})$ can be computed with Bayes' rule, and $\sum_d$ marginalizes out all of the task components of the states.

Once these computations are completed till the threshold ν is reached for each τ_k in the support of $\mathrm{p}(\tau_k)$, the distributions $\mathrm{p}(\tau_{k+1})$ and $\mathbf{g}_{\tau_{k+1}}$ must be truncated to a support of size K_τ and renormalized.

3.6.2 The Prior-To-Execution Update

Suppose that the current on-execution update is computing $\mathbf{g}_{\tau_{k+1}}$ and $\mathrm{p}(\tau_{k+1})$. Then for all tasks in the sequence with index $m > k$, there are dynamics (using Q_{ij} as discussed in section 3.4) that must be propagated forward during this update. Let d_{mk} be the task state of the task at position m in the sequence at time τ_k, i.e. at the start of task k. Then $\mathrm{p}(d_{m(k+1)}|\tau_{k+1})$ can be computed from $\mathrm{p}(d_{mk}|\tau_k)$ (matrix form $\mathbf{d}_{mk}$), as follows:

$$\begin{aligned} \mathbf{d}_{\tau_k}[0] &= \mathbf{d}_{mk} \\ \mathbf{d}_{\tau_k}[t+1] &= \mathbf{Q}^T \mathbf{f}_{\tau_k}[t] \end{aligned} \tag{3.11}$$

where $\mathbf{f}_{\tau_k}[t] = (\sum_d \mathbf{s}_{\tau_k}[t], \mathbf{d}_{\tau_k}[t])$ is obtained by marginalizing the task component out of $\mathbf{s}_{\tau_k}[t]$ and combining the resulting agent state $\mathbf{g}_{\tau_k}[t]$ distribution with $\mathbf{d}_{\tau_k}[t]$, $\mathbf{Q}$ is the prior-to-execution transition matrix for the m^{th} task,

This recursion allows the computation of the task state distribution

$$\mathbf{d}_{m(k+1)} = \sum_{\tau_k} \mathbf{d}_{\tau_k}[\tau_{k+1} - \tau_k]\mathrm{p}(\tau_k|\tau_{k+1}). \tag{3.12}$$

Finally, note that if the m^{th} task is independent of the agent prior to its execution, this simplifies considerably to

$$\mathbf{d}_{\tau_m} = \left(\mathbf{Q}^T\right)^{\tau_m} \mathbf{d}_{m0}. \tag{3.13}$$

3.7 Algorithm and Complexity Analysis

This task sequence evaluation scheme leads directly to the development of a task assignment algorithm based on a sequential greedy procedure. The MDP task allocation procedure in Algorithm 2 operates by inserting candidate tasks into the possible positions (denoted $\oplus_k$ in the algorithm block) along each agent's task sequence, evaluating those sequences, and selecting the best possible combination of agent/task/insertion index until all tasks have either been assigned or no longer increase the agent's expected reward. This process can be decentralized using an algorithm such as CBBA [24], which provides similar convergence and performance guarantees to centralized sequential greedy algorithms such as Algorithm 2. The inner loop task sequence evaluation procedure in Algorithm 3 is responsible for evaluating candidate sequences.

3.7.1 Policy Computation Complexity

As discussed earlier, policies must be precomputed for all $\bar{N}_a\bar{N}_t$ combinations of $\bar{N}_a$ unique agent types and $\bar{N}_t$ unique task types prior to running Algorithms 2 and 3. A relatively straightforward and well-known way to solve the planning problem for each of these combinations is to use value iteration with a tabular representation; in

this case, the complexity of finding an ϵ-optimal policy for all combinations is [102]

$$O\left(\bar{N}_a \bar{N}_t |\mathcal{S}|^2_{\max} |\mathcal{A}|_{\max} \frac{\log(1/\epsilon)}{(1-\gamma)}\right), \tag{3.14}$$

where $|\mathcal{S}|_{\max}$, $|\mathcal{A}|_{\max}$ are the maximum state and action space sizes for any combination. As this is a one-time precomputation, it does not affect the time to replan for new scenarios involving the same types of task and agent. Naturally, feature or basis function representations [103] and approximate methods [93] may be used during the computation of π_{ij} to reduce memory usage and computational complexity; this extends the scope of this approach to scenarios with a much larger $|\mathcal{S}|_{\max}$ than those which may be handled with value iteration and a tabular representation.

3.7.2 Allocation Complexity

The worst-case complexity of Algorithms 2 and 3 combined is polynomial in N_t and constant in N_a:

$$O\left(N_t^5 |\mathcal{S}|^2_{\max} K_\tau \left(\log \frac{1}{1-\nu_{th}}\right) \left(\log \frac{1}{|\lambda^*|}\right)^{-1}\right), \tag{3.15}$$

where $\lambda^* = \arg\max_{\lambda \in \text{eig}(\mathbf{P}):|\lambda|<1} |\lambda|$ is the eigenvalue of $\mathbf{P}$ with maximum magnitude less than 1. The N_t term arises from an $O(N_t^3)$ number of task insertions, with $O(N_t^2)$ on/prior-to-execution updates per insertion. The λ^* term captures the convergence rate of $\sum_{t=1}^{T} p(\tau_{k+1} = t + \tau_k | \tau_k)$ to a number larger than ν_{th} during on-execution updates.

3.8 Example: Target Identification

The MDP task allocation algorithm was tested on a multiagent multitarget surveillance scenario in a 10×10 gridworld environment with N_a agents and N_t targets. A basic version of this problem with 2 agents and 3 targets (the algorithm was tested on much larger problem sizes than this) is illustrated in Figure 3-1. In this example,

Algorithm 2 MDP Task Allocation

Input: Set of tasks $\mathcal{T}$, agents $\mathcal{Z}$

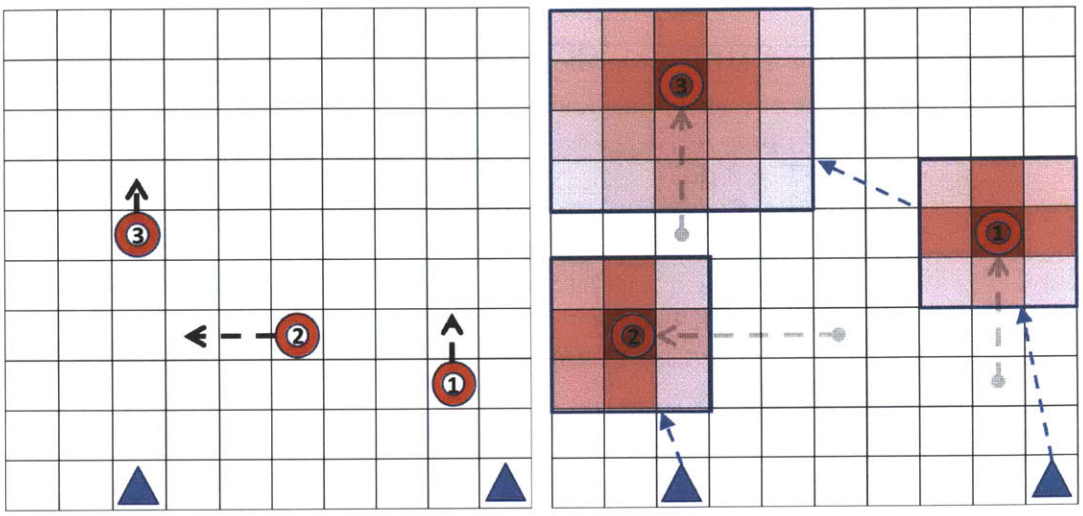
$\forall i \in \mathcal{Z} : \mathbf{p}_i \leftarrow \{\}, J_{i,\text{prev}} \leftarrow 0$
while $\mathcal{T} \neq \emptyset$ **do**
  for $i \in \mathcal{Z}, j \in \mathcal{T}, k \in \mathbf{p}_i$ **do**
    $J_{ijk} \leftarrow$EvalTaskSeq$(\mathbf{p}_i \oplus_k j)$
  end for
  $\hat{i}, \hat{j}, \hat{k} \leftarrow \arg\max J_{ijk} - J_{i,\text{prev}}$
  $\mathcal{T} \leftarrow \mathcal{T} \backslash \hat{j}$, $J_{\hat{i},\text{prev}} \leftarrow J_{\hat{i}\hat{j}\hat{k}}$, $\mathbf{p}_{\hat{i}} \leftarrow \mathbf{p}_{\hat{i}} \oplus_{\hat{k}} \hat{j}$
end while
return $\mathbf{p}_i \forall i \in \mathcal{Z}$

Algorithm 3 EvalTaskSeq

Input: $s_0, \tau_0, \mathbf{p}_i = \{j_0, \ldots, j_{n-1}\}$

$V \leftarrow 0$
for $k = 0 \rightarrow n-1$ **do**
  $V \leftarrow V + \mathbb{E}_{s_k,\tau_k}\left[\gamma^{\tau_k} V_{ij}^{\pi}(s_k)\right]$
  $\left(\mathbf{g}_{\tau_{k+1}}, \mathrm{p}(\tau_{k+1})\right) \leftarrow$OnExecUpdate$(\mathbf{s}_{\tau_k}, \mathrm{p}(\tau_k))$
  for $i = k+1 \rightarrow n-1$ **do**
    $\mathbf{d}_{\tau_i} \leftarrow$PriorExecUpdate$(\mathbf{d}_{\tau_i}, \mathbf{s}_{\tau_k}, \{\mathrm{p}(\tau_q)\}_{q=1}^{i})$
  end for
  $\mathbf{s}_{\tau_{k+1}} \leftarrow (\mathbf{g}_{\tau_{k+1}}, \mathbf{d}_{\tau_{k+1}})$
end for
return V

the goal for the fleet of agents was to capture images of as many targets as possible in the surveillance region. If an agent was in the same cell as a target it had an 80% chance of acquiring an image; if the target was in an adjacent cell of the agent it had a 20% chance of acquiring an image. Capturing an image of a target removed the target from the scenario and provided a reward of 10 to the agents, where rewards were discounted into the future using a factor of $\gamma = 0.9$. Targets moved stochastically with a dynamical model that was provided to the agents, and targets were able to leave the domain permanently, preventing agents from taking an image of them. Agents were capable of moving within a 5×5 square of cells around their current location, while targets were capable of moving within a 3×3 square of cells around their current location.

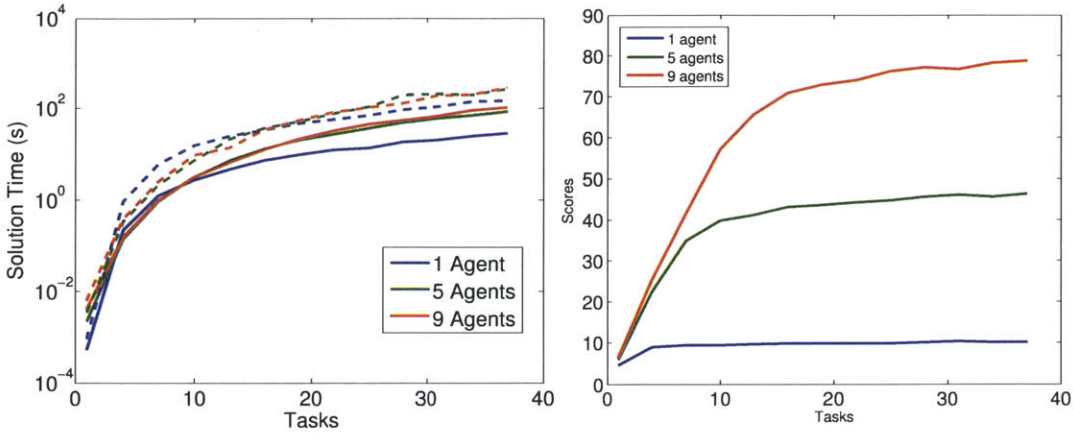


(a) Simple Scenario (b) Candidate Task Allocation

Figure 3-1: Simple scenario consisting of two agents (blue ▲) deciding how to pursue three targets (red ○), which highlights the effect of the growth in target position uncertainty over time. It is noted that the MDP task allocation algorithm addresses the full N_a agent and N_t task problem.

In this scenario, the multiagent MDP was transformed into a list of available tasks for the MDP task allocation algorithm by specifying each target as a single task. By specifying the tasks in this way, agents were able to explicitly coordinate with one another such that no two agents ever pursued the same target. Further, such tasks were guaranteed to satisfy the absorbing requirement for MDP tasks, as each target would eventually leave the scenario or be imaged by an agent.

This scenario was run in a Monte Carlo simulation consisting of 196 trials, each with randomized initial conditions, for every combination of $N_a = \{1, 5, 9\}$ and $N_t = \{1, \ldots, 36\}$. The main result of these tests is shown in Figures 3-2a-3-2b. Figure 3-2a demonstrates that the mean of the computation time for planning grows subexponentially with respect to the number of tasks, and generally remains constant with larger number of agents. The examination of Algorithm 3 refines this experimental estimate and yields a worst case quintic complexity with respect to the total number of tasks. Further, while it may seem counterintuitive that an increasing number of agents (going from 5-9) does not increase the solution time, this is caused by the fact



(a) (b)

Figure 3-2: (3-2a): Logarithmic plot of mean (solid) and worst-case (dashed) time performance as a function of the number of tasks. Both the mean and worst-case computational time scale polynomially. (3-2b) Plot of mission scores as a function of number of tasks.

that the most computationally expensive portion of the task allocation algorithm is the propagation of probability distributions. With an increased number of agents, these distributions are generally propagated through shorter task sequences, as each agent is required to do fewer tasks. It is noted that a full computation time for this algorithm would include the computation of policies for each of the MDP tasks; but as these are only computed once before the algorithm is run for each *type* of task, they do not affect the scaling of computation time with the number of agents or tasks. In the scenario run above the policy computation took about 3 seconds per task type. Figure 3-2b demonstrates that the mission score behaves intuitively as the number of agents and tasks are varied. With a fixed number of agents and increasing number of tasks, the score increases until it reaches an upper bound, illustrating the number of tasks at which the team becomes fully occupied. As the number of agents increases, that upper bound increases correspondingly.

If modelled as a multiagent MDP, this surveillance scenario has a state-action space size of approximately $X^{2N_t+2N_a}Y^{2N_a}$ (this is approximate because targets have an extra state for having left the domain, and agents have some movement actions restricted in certain grid cells). For example, in the largest surveillance scenario tested

during the Monte Carlo simulation, this corresponds to a state-action space of cardinality $\sim 10^{102}$. This exponential complexity in the number of agents and tasks is well known [21] in decision-making problems involving uncertainty, and causes solutions to such problems to become computationally intractable as the number of agents and tasks increases. Recent work has made efforts to reduce this complexity [104]; however, the complexity only becomes polynomial given certain key assumptions, and in the general case it remains exponential. As demonstrated in Figure 3-2a, the present MDP task allocation framework mitigates this issue and yields a complexity which is polynomial for general stochastic multiagent decision-making scenarios.

3.9 Summary

This work developed a multiagent task allocation algorithm for general stochastic tasks based on Markov decision processes. This algorithm is scalable to situations involving a large number of agents and tasks, while capable of incorporating the generality of MDP models of those tasks. Results showed the feasibility of using these methods in realistic mission scenarios with a large number of agents and tasks.

Chapter 4

The Dynamic Means Algorithm

4.1 Overview

A key assumption in MDP task allocation is that the models of the MDP tasks are well-known; this assumption is required both in finding the policies for each task, and in propagating the state distributions along task paths for each agent to evaluate bundle scores. To remove this assumption, a model learning procedure that is flexible enough to capture an unknown number of task types (e.g. for a target tracking task, different target behaviors), but fast enough to be run in real-time (in order to quickly model/identify different types of task while they are being performed) is required.

Parametric modeling techniques are not well-suited to this endeavor, as they cannot capture the different task types that exist without knowing how many exist a priori. Bayesian nonparametrics, on the other hand, possess this flexibility and are well-suited for task model learning; in particular, because the model learning problem involves an unknown number of task types, a Dirichlet process-based model is a natural choice. Of this class of models, the dependent Dirichlet process (DDP) is perhaps the best candidate: It captures an unknown number of evolving models, which allows it to adapt to changing environments; and inference algorithms for it do not grow in complexity over time (despite streaming data) due to periodic compression of data into sufficient statistics.

However, inference procedures for the DDP do not operate on the timescales

required for real-time multiagent planning [105]. Thus, this chapter presents a novel algorithm, based upon the dependent Dirichlet process mixture model (DDPMM), for clustering batch-sequential data containing an unknown number of temporally evolving clusters. The algorithm is derived via a low-variance asymptotic analysis of the Gibbs sampling algorithm for the DDPMM, and provides a hard clustering with convergence guarantees similar to those of the k-means algorithm. Empirical results from a synthetic test with moving Gaussian clusters and a test with real ADS-B aircraft trajectory data demonstrate that the algorithm provides a clustering accuracy comparable to other similar algorithms, while posing a significantly lower computational burden.

This chapter is based on the paper "Dynamic Clustering via Asymptotics of the Dependent Dirichlet Process Mixture" [106], which was submitted to the 2013 Neural Information Processing Systems Conference in collaboration with M. Liu, B. Kulis, and J. How.

4.2 Introduction

The Dirichlet process mixture model (DPMM) is a powerful tool for clustering data that enables the inference of an unbounded number of mixture components, and has been widely studied in the machine learning and statistics communities [17, 19, 49, 50]. Despite its flexibility, it assumes the observations are exchangeable, and therefore that the data points have no inherent ordering that influences their labeling. This assumption is invalid for modeling temporally/spatially evolving phenomena, in which the order of the data points plays a principal role in creating meaningful clusters. The dependent Dirichlet process (DDP), originally formulated by MacEachern [48], provides a prior over such evolving mixture models, and is a promising tool for incrementally monitoring the dynamic evolution of the cluster structure within a dataset. More recently, a construction of the DDP built upon completely random measures [12] led to the development of the dependent Dirichlet process Mixture model (DDPMM) and a corresponding approximate posterior inference Gibbs sampling algorithm. This

model generalizes the DPMM by including birth, death and transition processes for the clusters in the model.

The DDPMM is a Bayesian nonparametric (BNP) model, part of an ever-growing class of probabilistic models for which inference captures uncertainty in the both the number of parameters and their values. While these models are powerful in their capability to capture complex structures in data without requiring explicit model selection, they suffer some practical shortcomings. Inference techniques for BNPs typically fall into two classes: sampling methods (e.g. Gibbs sampling [50] or particle learning [19]) and optimization methods (e.g. variational inference [17] or stochastic variational inference [18]). Current methods based on sampling do not scale well with the size of the dataset [52]. Most optimization methods require analytic derivatives and the selection of an upper bound on the number of clusters a priori, where the computational complexity increases with that upper bound [17, 18]. State-of-the-art techniques in both classes are not ideal for use in contexts where performing inference quickly and reliably on large volumes of streaming data is crucial for timely decision-making, such as autonomous robotic systems [53–55]. On the other hand, many classical clustering methods [56–58] scale well with the size of the dataset and are easy to implement, and advances have recently been made to capture the flexibility of Bayesian nonparametrics in such approaches [59]. However, as of yet there is no classical algorithm that captures all the processes (cluster birth, death, and movement) that the DDP mixture can capture within data posessing a dynamic cluster structure.

This chapter discusses the Dynamic Means algorithm, a novel hard clustering algorithm for spatio-temporal data derived from the low-variance asymptotic limit of the Gibbs sampling algorithm for the dependent Dirichlet process Gaussian mixture model. This algorithm captures the scalability and ease of implementation of classical clustering methods, along with the representational power of the DDP prior, and is guaranteed to converge to a local minimum of a k-means-like cost function. The algorithm is significantly more computationally tractable than Gibbs sampling, particle learning, and variational inference for the DDP mixture model in practice,

while providing equivalent or better clustering accuracy on the examples presented. The performance and characteristics of the algorithm are demonstrated in a test on synthetic data, with a comparison to those of Gibbs sampling, particle learning and variational inference. Finally, the applicability of the algorithm to real data is presented through an example of clustering a spatio-temporal dataset of aircraft trajectories recorded across the United States.

4.3 Asymptotic Analysis of the DDP Mixture

The dependent Dirichlet process Gaussian mixture model (DDP-GMM) serves as the foundation upon which the present work is built. The generative model of a DDP-GMM at time step t is

$$\begin{aligned} \{\theta_{kt}, \pi_{kt}\}_{k=1}^{\infty} &\sim \mathrm{DP}(\mu_t) \\ \{z_{it}\}_{i=1}^{N_t} &\sim \mathrm{Categorical}(\{\pi_{kt}\}_{k=1}^{\infty}) \\ \{y_{it}\}_{i=1}^{N_t} &\sim \mathcal{N}(\theta_{z_{it}t}, \sigma I) \end{aligned} \tag{4.1}$$

where θ_{kt} is the mean of cluster k, π_{kt} is the categorical weight for class k, y_{it} is a d-dimensional observation vector, z_{it} is a cluster label for observation i, and μ_t is the base measure from (2.12). Throughout the rest of this chapter, the subscript kt refers to quantities related to cluster k at time step t, and subscript it refers to quantities related to observation i at time step t.

The Gibbs sampling algorithm for the DDP-GMM iterates between sampling labels z_{it} for datapoints y_{it} given the set of parameters $\{\theta_{kt}\}$, and sampling parameters θ_{kt} given each group of data $\{y_{it} : z_{it} = k\}$. Assuming the transition model T is Gaussian, and the subsampling function q is constant, the functions and distributions used in the Gibbs sampling algorithm are: the prior over cluster parameters, $\theta \sim \mathcal{N}(\phi, \rho I)$; the likelihood of an observation given its cluster parameter, $y_{it} \sim \mathcal{N}(\theta_{kt}, \sigma I)$; the distribution over the transitioned cluster parameter given its last known location after Δt_k time steps, $\theta_{kt} \sim \mathcal{N}(\theta_{k(t-\Delta t_k)}, \xi \Delta t_k I)$; and the sub-

sampling function $q(\theta) = q \in (0, 1)$. Given these functions and distributions, the low-variance asymptotic limits (i.e. $\sigma \to 0$) of these two steps are discussed in the following sections.

4.3.1 Setting Labels Given Parameters

In the label sampling step, a datapoint y_{it} can either create a new cluster, join a current cluster, or revive an old, transitioned cluster. Using the distributions defined previously, the label assignment probabilities are

$$p(z_{it} = k | \dots) \propto \begin{cases} \alpha_t (2\pi(\sigma+\rho))^{-d/2} \exp\left(-\dfrac{||y_{it} - \phi||^2}{2(\sigma+\rho)}\right) & k = K+1 \\ (c_{kt} + n_{kt})(2\pi\sigma)^{-d/2} \exp\left(-\dfrac{||y_{it} - \theta_{kt}||^2}{2\sigma}\right) & n_{kt} > 0 \\ q_{kt} c_{kt} (2\pi(\sigma + \xi\Delta t_k))^{-d/2} \exp\left(-\dfrac{||y_{it} - \theta_{k(t-\Delta t_k)}||^2}{2(\sigma+\xi\Delta t_k)}\right) & n_{kt} = 0 \end{cases}$$

where $q_{kt} = q^{\Delta t_k}$ due to the fact that $q(\theta)$ is constant over Ω, and $\alpha_t = \alpha_\nu \dfrac{1-q^t}{1-q}$, where α_ν is the concentration parameter for the innovation process, F_t. The low-variance asymptotic limit of this label assignment step yields meaningful assignments as long as α_ν, ξ, and q vary appropriately with σ; thus, setting α_ν, ξ, and q as follows (where λ, τ and Q are positive constants):

$$\alpha_\nu = (1 + \rho/\sigma)^{d/2} \exp\left(-\frac{\lambda}{2\sigma}\right), \quad \xi = \tau\sigma, \quad q = \exp\left(-\frac{Q}{2\sigma}\right) \tag{4.2}$$

yields the following assignments in the limit as $\sigma \to 0$:

$$z_{it} = \underset{k}{\operatorname{argmin}} \{J_k\}, \tag{4.3}$$

$$\text{where} \quad J_k = \begin{cases} ||y_{it} - \theta_{kt}||^2 & \text{if } \theta_k \text{ instantiated} \\ Q\Delta t_k + \dfrac{||y_{it} - \theta_{k(t-\Delta t_k)}||^2}{\tau\Delta t_k + 1} & \text{if } \theta_k \text{ old, uninstantiated} \\ \lambda & \text{if } \theta_k \text{ new} \end{cases} \tag{4.4}$$

In this assignment step, $Q\Delta t_k$ acts as a cost penalty for reviving old clusters that increases with the time since the cluster was last seen, $\tau\Delta t_k$ acts as a cost reduction to account for the possible motion of clusters since they were last instantiated, and λ acts as a cost penalty for introducing a new cluster.

4.3.2 Setting Parameters given Labels

In the parameter sampling step, the parameters are sampled using the distribution

$$\mathrm{p}(\theta_{kt}|\{y_{it} : z_{it} = k\}) \propto p(\{y_{it} : z_{it} = k\}|\theta_{kt})\mathrm{p}(\theta_{kt}) \tag{4.5}$$

There are two cases to consider when setting a parameter θ_{kt}. Either $\Delta t_k = 0$ and the cluster is new in the current time step, or $\Delta t_k > 0$ and the cluster was previously created, disappeared for some amount of time, and then was revived in the current time step.

New Cluster Suppose cluster k is being newly created. In this case, $\theta_{kt} \sim \mathcal{N}(\phi, \rho)$. Using the fact that a normal prior is conjugate a normal likelihood, we have a closed-form posterior for θ_{kt}:

$$\begin{gathered} \theta_{kt}|\{y_{it} : z_{it} = k\} \sim \mathcal{N}\left(\theta_{\text{post}}, \sigma_{\text{post}}\right) \\ \theta_{\text{post}} = \sigma_{\text{post}}\left(\frac{\phi}{\rho} + \frac{\sum_{i=1}^{n_{kt}} y_{it}}{\sigma}\right), \quad \sigma_{\text{post}} = \left(\frac{1}{\rho} + \frac{n_{kt}}{\sigma}\right)^{-1} \end{gathered} \tag{4.6}$$

Then letting $\sigma \to 0$ yields

$$\theta_{kt} = \frac{\left(\sum_{i=1}^{n_{kt}} y_{it}\right)}{n_{kt}} \stackrel{\text{def}}{=} m_{kt} \tag{4.7}$$

where n_{kt} is the number of observations in cluster k in the current timestep, and m_{kt} is the mean of the observations in cluster k in the current timestep.

Revived Cluster Suppose there are Δt_k time steps where cluster k was not observed, but there are now n_{kt} data points with mean m_{kt} assigned to it in this time

step. In this case,

$$\mathrm{p}(\theta_{kt}) = \int_{\theta} T(\theta_{kt}|\theta)\mathrm{p}(\theta)\,\mathrm{d}\theta,\ \theta \sim \mathcal{N}(\theta', \sigma'). \tag{4.8}$$

Again using conjugacy of normal likelihoods and priors,

$$\begin{gathered} \theta_{kt}|\{y_{it} : z_{it} = k\} \sim \mathcal{N}\left(\theta_{\text{post}}, \sigma_{\text{post}}\right) \\ \theta_{\text{post}} = \sigma_{\text{post}}\left(\frac{\theta'}{\xi\Delta t_k + \sigma'} + \frac{\sum_{i=1}^{n_{kt}} y_{it}}{\sigma}\right),\ \sigma_{\text{post}} = \left(\frac{1}{\xi\Delta t_k + \sigma'} + \frac{n_{kt}}{\sigma}\right)^{-1} \end{gathered} \tag{4.9}$$

Similarly to the label assignment step, let $\xi = \tau\sigma$. Then as long as $\sigma' = \sigma/w$, $w > 0$ (which holds if (4.9) is used to recursively keep track of the parameter posterior), taking the asymptotic limit of this as $\sigma \to 0$ yields:

$$\theta_{kt} = \frac{\theta'(w^{-1} + \Delta t_k\tau)^{-1} + n_{kt}m_{kt}}{(w^{-1} + \Delta t_k\tau)^{-1} + n_{kt}} \tag{4.10}$$

that is to say, the revived θ_{kt} is a weighted average of estimates using current timestep data and previous timestep data. τ controls how much the current data is favored – as τ increases, the weight on current data increases, which is explained by the fact that the uncertainty in where the old θ' transitioned to increases with τ. It is also noted that if $\tau = 0$, this reduces to a simple weighted average using the amount of data collected as weights. This makes sense, as the $\tau = 0$ case implies that there is no movement in the cluster centres, and so no reduction in old parameter weight occurs (equivalently, all past information collected is retained).

Combined Update Combining the updates for new cluster parameters and old transitioned cluster parameters yields a recursive update scheme:

$$\begin{aligned} \theta_{k0} &= m_{k0} \\ w_{k0} &= n_{k0} \end{aligned} \qquad \begin{aligned} \gamma_{kt} &= \left((w_{k(t-\Delta t_k)})^{-1} + \Delta t_k\tau\right)^{-1} \\ \theta_{kt} &= \frac{\theta_{k(t-\Delta t_k)}\gamma_{kt} + n_{kt}m_{kt}}{\gamma_{kt} + n_{kt}} \\ w_{kt} &= \gamma_{kt} + n_{kt} \end{aligned} \tag{4.11}$$

where time step 0 here corresponds to when the cluster is first created. An interesting interpretation of this update is that it behaves like a standard Kalman filter, in which w_{kt}^{-1} serves as the current estimate variance, τ serves as the process noise variance, and n_{kt} serves as the inverse of the measurement variance.

4.4 The Dynamic Means Algorithm

In this section, some further notation is introduced for brevity:

$$\begin{aligned} \mathcal{Y}_t = \{y_{it}\}_{i=1}^{N_t}, \quad \mathcal{Z}_t = \{z_{it}\}_{i=1}^{N_t} \\ \mathcal{K}_t = \{(\theta_{kt}, w_{kt}) : n_{kt} > 0\}, \quad \mathcal{C}_t = \{(\Delta t_k, \theta_{k(t-\Delta t_k)}, w_{k(t-\Delta t_k)})\} \end{aligned} \tag{4.12}$$

where $\mathcal{Y}_t$ and $\mathcal{Z}_t$ are the sets of observations and labels at time step t, $\mathcal{K}_t$ is the set of currently active clusters (some are new with $\Delta t_k = 0$, and some are revived with $\Delta t_k > 0$), and $\mathcal{C}_t$ is the set of old cluster information.

4.4.1 Algorithm Description

As shown in the previous section, the low-variance asymptotic limit of the DDP Gibbs sampling algorithm is a deterministic observation label update (4.4) followed by a deterministic, weighted least-squares parameter update (4.11). Inspired by the original k-means[56] algorithm, applying these two updates iteratively yields an algorithm which clusters a set of observations at a single time step given cluster means and weights from past time steps (Algorithm 4). Applying Algorithm 4 to a sequence of batches of data yields a clustering procedure that is able to track a set of dynamically evolving clusters (Algorithm 5), and allows new clusters to emerge and old clusters to be forgotten. While this is the primary application of Algorithm 5, the sequence of batches need not be a temporal sequence. For example, Algorithm 5 may be used as an any-time clustering algorithm for large datasets, where the sequence of batches is generated by selecting random subsets of the full dataset.

The AssignParams function is exactly the update from (4.11) applied to each

Algorithm 4 CLUSTER

Input: $\mathcal{Y}_t$, $\mathcal{C}_t$, Q, λ, τ
  $\mathcal{K}_t \leftarrow \emptyset$, $\mathcal{Z}_t \leftarrow \emptyset$, $L_0 \leftarrow \infty$
  for $n = 1 \rightarrow \infty$ **do**
    $(\mathcal{Z}_t, \mathcal{K}_t) \leftarrow$ASSIGNLABELS$(\mathcal{Y}_t, \mathcal{Z}_t, \mathcal{K}_t, \mathcal{C}_t)$
    $(\mathcal{K}_t, L_n) \leftarrow$ASSIGNPARAMS$(\mathcal{Y}_t, \mathcal{Z}_t, \mathcal{C}_t)$
    if $L_n = L_{n-1}$ **then**
      return $\mathcal{K}_t$, $\mathcal{Z}_t$, L_n
    end if
  end for

$k \in \mathcal{K}_t$. Similarly, the ASSIGNLABELS function applies the update from (4.4) to each observation; however, in the case that a new cluster is created or an old one is revived by an observation, ASSIGNLABELS also creates a parameter for that new cluster based on the parameter update (4.11) with that single observation. The UPDATEC function is run after clustering observations from each time step, and constructs $\mathcal{C}_{t+1}$ by setting $\Delta t_k = 1$ for any new or revived cluster, and by incrementing Δt_k for any old cluster that was not revived:

$$\mathcal{C}_{t+1} = \{(\Delta t_k + 1, \theta_{k(t-\Delta t_k)}, w_{k(t-\Delta t_k)}) : k \in \mathcal{C}_t, k \notin \mathcal{K}_t\} \cup \{(1, \theta_{kt}, w_{kt}) : k \in \mathcal{K}_t\} \quad (4.13)$$

An important question is whether this algorithm is guaranteed to converge while clustering data in each time step. Indeed, it is; Theorem 4.4.1 shows that a particular cost function L_t monotonically decreases under the label and parameter updates (4.4) and (4.11) at each time step. Since $L_t \geq 0$, and it is monotonically decreased by Algorithm 4, the algorithm converges in a finite number of iterations by Corollary 4.4.2. Note that the Dynamic Means is only guaranteed to converge to a local optimum, similarly to the k-means [56] and DP-Means [59] algorithms.

Theorem 4.4.1 *Each iteration in Algorithm 4 monotonically decreases the cost func-*

Algorithm 5 Dynamic Means

Input: $\{\mathcal{Y}_t\}_{t=1}^{t_f}$, Q, λ, τ
  $\mathcal{C}_1 \leftarrow \emptyset$
  for $t = 1 \rightarrow t_f$ **do**
    $(\mathcal{K}_t, \mathcal{Z}_t, L_t) \leftarrow$CLUSTER$(\mathcal{Y}_t, \mathcal{C}_t, Q, \lambda, \tau)$
    $\mathcal{C}_{t+1} \leftarrow$UPDATEC$(\mathcal{Z}_t, \mathcal{K}_t, \mathcal{C}_t)$
  end for
  return $\{\mathcal{K}_t, \mathcal{Z}_t, L_t\}_{t=1}^{t_f}$

tion L_t, *where*

$$L_t = \sum_{k \in \mathcal{K}_t} \left(\overbrace{\lambda \left[\Delta t_k = 0\right]}^{\textit{New Cost}} + \overbrace{Q \Delta t_k}^{\textit{Revival Cost}} + \overbrace{\gamma_{kt} ||\theta_{kt} - \theta_{k(t-\Delta t_k)}||_2^2 + \sum_{\substack{y_{it} \in \mathcal{Y}_t \\ z_{it}=k}} ||y_{it} - \theta_{kt}||_2^2}^{\textit{Weighted-Prior Sum-Squares Cost}} \right) \tag{4.14}$$

Proof This proof proceeds by showing that each step of Dynamic Means (parameter assignment and label assignment) individually do not increase L_t. Thus, when these steps are iterated, L_t is monotonically decreasing.

The first step considered is the parameter update. During this update, the portion of L_t attributed to λ and Q does not change, as the cluster labels are held constant. Therefore, the only portion of L_t that needs to be considered is the weighted sum square cost. Taking the derivative of this with respect to θ_{kt}, and setting it equal to 0 yields

$$0 = \gamma_{kt}(\theta_{kt} - \theta_{k(t-\Delta t_k)}) + \sum_{z_{it}=k} (\theta_{kt} - y_{it}) \tag{4.15}$$

$$(\gamma_{kt} + n_{kt})\theta_{kt} = \gamma_{kt}\theta_{k(t-\Delta t_k)} + \sum_{z_{it}=k} y_{it} \tag{4.16}$$

$$\theta_{kt} = \frac{\gamma_{kt}\theta_{k(t-\Delta t_k)} + n_{kt} m_{kt}}{\gamma_{kt} + n_{kt}} \tag{4.17}$$

and since this portion of the cost function is convex in θ_{kt}, this is the minimizing θ_{kt} for cluster k, and thus this portion of the cost function is reduced. Note that if k is a newly created cluster in timestep t, $\gamma_{kt} = 0$ and thus θ_{kt} is set to the current sample

mean, m_{kt}.

The second step considered is the label update. Here, there are a number of cases: 1) the update moves an observation from a currently active cluster to another currently active cluster; 2) the update creates a new cluster; 3) the update revives an old cluster; and 4) the observation was the last in its group, and the cluster is destroyed. Each case will be considered separately. Note that the amount that an observation y in cluster k contributes to the cost function prior to reassignment is $\|y - \theta_{kt}\|^2$.

1. Suppose the observation y in cluster k becomes assigned to a currently active cluster j. Since the cluster was already active, the observation now contributes $\|y - \theta_{jt}\|^2$ to the cost, and $\|y - \theta_{jt}\|^2 \leq \|y - \theta_{kt}\|^2$ because the label assignment step chose the minimum of these two. Thus, the cost does not increase.

2. Suppose the observation y in cluster k is assigned to a brand new cluster K with cost λ (the new $\|y - \theta_{Kt}\|^2$ cost is 0, since θ_{Kt} is set to $\theta_{Kt} = y$ when a new cluster is created). Then we know that $\lambda + \|y - \theta_{Kt}\|^2 = \lambda \leq \|y - \theta_{kt}\|^2$ due to the label assignment step choosing the minimum of these two, and thus the cost does not increase.

3. Suppose the observation y in cluster k revives an old cluster j. Then

$$\frac{||y - \theta_{j(t-\Delta t_k)}||_2^2}{\Delta t_j \tau + 1} + Q\Delta t_j \leq ||y - \theta_{kt}||_2^2 \tag{4.18}$$

and further, using the parameter update (4.11),

$$\theta_{jt} = \frac{\theta_{j(t-\Delta t_k)}\gamma_{jt} + y}{\gamma_{jt} + 1} \tag{4.19}$$

rearranging the above yields two relations:

$$||\theta_{jt} - y||_2^2 = \frac{\gamma_{jt}^2}{(1 + \gamma_{jt})^2}||\theta_{j(t-\Delta t_j)} - y||_2^2 \tag{4.20}$$

$$||\theta_{jt} - y||_2^2 = \gamma_{jt}^2||\theta_{jt} - \theta_{j(t-\Delta t_j)}||_2^2 \tag{4.21}$$

where $\gamma_{jt} = \left((w_{j(t-\Delta t_j)})^{-1} + \Delta t_j \tau\right)^{-1}$. Placing these terms in a weighted least-squares-like configuration yields

$$||\theta_{jt} - y||_2^2 + \gamma_{jt}||\theta_{jt} - \theta_{j(t-\Delta t_j)}||_2^2 = (1 + \gamma_{jt}^{-1})||\theta_{jt} - y||_2^2 \tag{4.22}$$

$$= \frac{(1 + \gamma_{jt}^{-1})\gamma_{jt}^2}{(1 + \gamma_{jt})^2}||\theta_{j(t-\Delta t_j)} - y||_2^2 \tag{4.23}$$

$$= \frac{1}{1 + \gamma_{jt}^{-1}}||\theta_{j(t-\Delta t_j)} - y||_2^2 \tag{4.24}$$

and subbing in for the γ_{jt} term,

$$||\theta_{jt} - y||_2^2 + \gamma_{jt}||\theta_{jt} - \theta_{j(t-\Delta t_j)}||_2^2 = \frac{1}{1 + (w_{j(t-\Delta t_j)})^{-1} + \Delta t_j \tau}||\theta_{j(t-\Delta t_j)} - y||_2^2 \tag{4.25}$$

$$\leq \frac{||\theta_{j(t-\Delta t_j)} - y||_2^2}{1 + \Delta t_j \tau} \tag{4.26}$$

Therefore, if the DDP k-means algorithm instantiates an old cluster,

$$||\theta_{jt} - y||_2^2 + \gamma_{jt}||\theta_{jt} - \theta_{j(t-\Delta t_j)}||_2^2 + Q\Delta t_j \leq \frac{||y - \theta_{j(t-\Delta t_k)}||_2^2}{\Delta t_j \tau + 1} + Q\Delta t_j \tag{4.27}$$

$$\leq ||y - \theta_{kt}||_2^2 \tag{4.28}$$

and the new contribution to the cost is less than the old contribution.

4. Suppose the observation y in cluster k was the last observation in its cluster prior to reassignment, and y became assigned to a different cluster. If y was previously part of a newly created cluster in this time step, the cost decreases by λ after y is reassigned; likewise, if y was previously part of a revived cluster, the cost decreases by $Q\Delta t_k + \gamma_{kt}||\theta_{kt} - \theta_{k(t-\Delta t_j)}||^2$. Thus, the removal of a cluster that becomes empty after reassignment does not increase the cost.

■

Corollary 4.4.2 *Algorithm 4 terminates at a local cost optimum after a finite number of iterations.*

Proof This result is trivial given Theorem 4.4.1. Since L_t is monotonically decreasing, and bounded below by 0, L_t converges. Further, there are a finite number of possible data labellings. Thus, after a finite number of iterations, the labelling reaches a fixed point, and the algorithm terminates.

The cost function is comprised of a number of components for each currently active cluster $k \in \mathcal{K}_t$: A penalty for new clusters based on λ, a penalty for old clusters based on Q and Δt_k, and finally a prior-weighted sum of squared distance cost for all the observations in cluster k. It is noted that for new clusters, $\theta_{kt} = \theta_{k(t-\Delta t_k)}$ since $\Delta t_k = 0$, so the least squares cost is unweighted. The AssignParams function calculates this cost function in each iteration of Algorithm 4, and the algorithm terminates once the cost function does not decrease during an iteration.

4.4.2 Reparameterizing the Algorithm

In order to use the dynamic means algorithm, there are three free parameters to select: λ, Q, and τ. While λ represents how far an observation can be from a cluster before it is placed in a new cluster, and thus can be tuned intuitively, Q and τ are not so straightforward. The parameter Q represents a conceptual added distance from any data point to a cluster for every time step that the cluster is not observed. The parameter τ represents a conceptual reduction of distance from any data point to a cluster for every time step that the cluster is not observed. How these two quantities affect the algorithm, and how they interact with the setting of λ, is hard to judge.

Instead of picking Q and τ directly, the algorithm may be reparameterized by picking $N_Q, k_\tau \in \mathbb{R}_+$, $N_Q > 1$, $k_\tau \geq 1$, and given a choice of λ, setting

$$Q = \lambda / N_Q \quad \tau = \frac{N_Q(k_\tau - 1) + 1}{N_Q - 1}. \tag{4.29}$$

If Q and τ are set in this manner, N_Q represents the number (possibly fractional) of time steps a cluster can be unobserved before the label update (4.4) will never revive that cluster, and $k_\tau \lambda$ represents the maximum squared distance away from a cluster center such that after a single time step, the label update (4.4) will revive

that cluster. As N_Q and k_τ are specified in terms of concrete algorithmic behavior, they are intuitively easier to set than Q and τ.

4.5 Applications

4.5.1 Synthetic Gaussian Motion Data

In this experiment, moving Gaussian clusters on $[0,1] \times [0,1]$ were generated synthetically over a period of 100 time steps. In each step, there was some number of clusters, each having 15 data points. The data points were sampled from a symmetric Gaussian distribution with a standard deviation of 0.05. Between time steps, the cluster centers moved randomly, with displacements sampled from the same distribution. At each time step, each cluster had a 0.05 probability of being destroyed.

This data was clustered with Dynamic Means (with 3 random assignment ordering restarts), DDP-GMM Gibbs sampling, variational inference, and particle learning on a computer with an Intel i7 processor and 16GB of memory. First, the number of clusters was fixed to 5, and the parameter space of each algorithm was searched for the best possible performance in terms of cluster label accuracy (taking into account label consistency across time steps). For the Dynamic Means algorithm the space $(\lambda, T_Q, k_\tau) \in [0, 0.16] \times [0, 10] \times [1, 6]$ was tested, while for the comparison algorithms the space $(\log_{10}(\alpha), q) \in [-4, 2] \times [0, 1]$ was tested, with 50 trials at each parameter setting. The results of this parameter sweep for the Dynamic Means algorithm are shown in Figures 4-1a - 4-1c. Figures 4-1a and 4-1b show how the average clustering accuracy varies with the parameters after fixing either k_τ or T_Q to their values at the maximum accuracy parameter setting over the full space. The Dynamic Means algorithm had a similar robustness with respect to variations in its parameters as the comparison algorithms, when considering parameter deviations in terms of percentages of the maximum accuracy parameter setting. The histogram in Figure 4-1c demonstrates that the clustering speed is robust to the setting of parameters.

Using the best parameter setting with respect to mean cluster label accuracy for

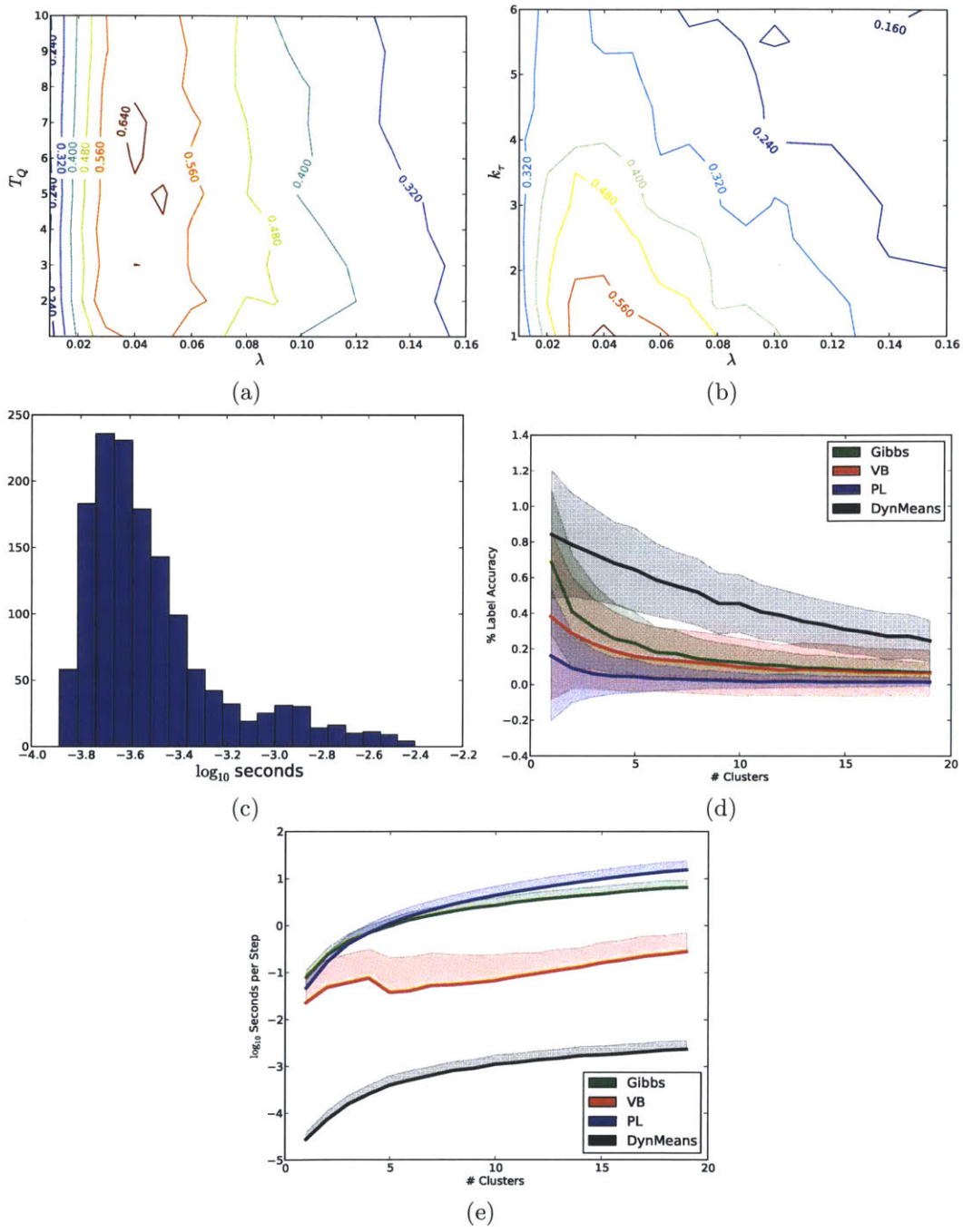


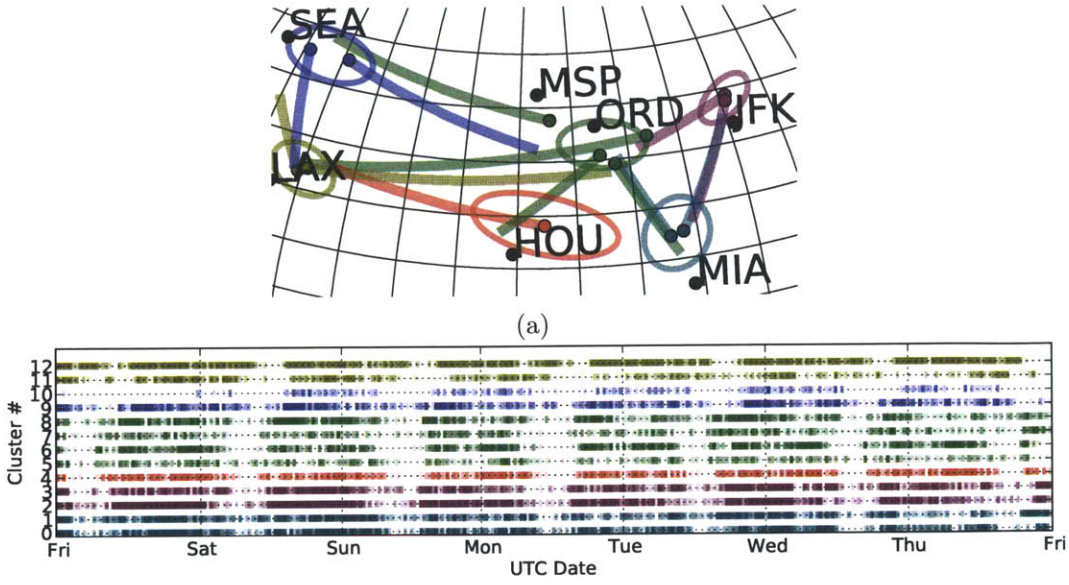
Figure 4-1: (4-1a - 4-1c): Accuracy contours and CPU time histogram for the Dynamic Means algorithm. (4-1d - 4-1e): Comparison with Gibbs sampling, variational inference, and particle learning. Shaded region indicates 1σ interval; in (4-1e), only upper half is shown.

each algorithm, the data as described above were clustered in 50 trials with a varying number of clusters present in the data. For the Dynamic Means algorithm, parameter values $\lambda = 0.04$, $T_Q = 6.8$, and $k_\tau = 1.01$ were used, and the algorithm was again given 3 attempts with random labelling assignment orders, where the lowest cost solution of the 3 was picked to proceed to the next time step. For the other algorithms, the parameter values $\alpha = 1$ and $q = 0.05$ were used, with a Gaussian transition distribution variance of 0.05. The number of samples for the Gibbs sampling algorithm was 5000 with one recorded for every 5 samples, the number of particles for the particle learning algorithm was 100, and the variational inference algorithm was run to a tolerance of 10^{-20} with the maximum number of iterations set to 5000.

In Figures 4-1d and 4-1e, the labelling accuracy and clustering time (respectively) for the algorithms is shown. Note that the sampling algorithms (Gibbs and PL) were handicapped to generate Figure 4-1d; the best posterior sample in terms of labelling accuracy was selected at each time step, which required knowledge of the true labelling. This example demonstrates that the Dynamic Means algorithm outperforms other standard inference algorithms in terms of label accuracy, while having the benefit of fast convergence to a solution. Note that the label accuracy computation included enforcing consistency across timesteps, to allow tracking individual cluster trajectories. If this is not enforced (i.e. accuracy considers each time step independently), the other algorithms provide accuracies more comparable to those of the Dynamic Means algorithm.

4.5.2 Aircraft Trajectory Clustering

In persistent surveillance missions, the ability to autonomously recognize anomalous situations and behaviors allows the intelligent allocation of resources to best address those anomalies. In the case of monitoring mobile agent trajectories, characterizing typical spatial and temporal patterns in their motions is an important step towards the detection of such anomalies. In this experiment, the Dynamic Means algorithm was used to find the typical spatial and temporal patterns in the motions of commercial aircraft, a domain in which anomaly detection is safety-critical.



(a)

(b)

Figure 4-2: Results of the GP aircraft trajectory clustering. (4-2a): A map (labelled with major US city airports) showing the overall aircraft flows for 12 trajectories, with colors and 1σ confidence ellipses corresponding to takeoff region (multiple clusters per takeoff region), colored dots indicating mean takeoff position for each cluster, and lines indicating the mean trajectory for each cluster. (4-2b): A track of plane counts for the 12 clusters during the week, with colour intensity proportional to the number of takeoffs at each time.

Automatic dependent surveillance-broadcast (ADS-B) data, including plane identification, timestamp, latitude, longitude, heading and speed, was collected from all transmitting planes across the United States during the week from 2013-3-22 1:30:0 to 2013-3-28 12:0:0 UTC. The following procedure was used to create feature vectors for clustering the data. First, individual ADS-B messages were connected together based on their plane identification and timestamp to form trajectories, and erroneous trajectories were filtered based on reasonable spatial/temporal bounds, yielding 17,895 unique trajectories. Then, for each trajectory, a Gaussian process was trained using the latitude and longitude of each ADS-B point along the trajectory as the inputs and the North and East components of plane velocity at those points as the outputs. Next, the mean velocity from the Gaussian process was queried for each point on a

regular lattice across the USA with 10 latitudes and 20 longitudes, and used to create a 400-dimensional feature vector for each trajectory. Finally, the resulting 17,895 feature vectors were clustered using the Dynamic Means algorithm, DP-Means algorithm (run on the entire dataset in a single batch), and the DDPGMM Gibbs sampling algorithm (50 sampling iterations total per time step) for all plane take-offs occuring in a sequence of half hour time windows.

The clustering results of this exercise from the Dynamic Means algorithm are shown in Figure 4-2. Over 10 trials, the Dynamic Means algorithm took an average of 4.9s to process all the feature vectors, the DP-Means algorithm took an average of 72.3s, while the DDP Gibbs sampling algorithm took an average of 5426.1s. The DP-Means algorithm provided a model with about twice as many clusters (mostly near duplicates of other clusters) due to the inability to capture slight variations through cluster transition processes and the inability to remove erroneous clusters by modeling cluster death processes. The DDP Gibbs sampling algorithm did not give reliable results and proved to be very difficult to tune on this dataset, due to the high dimensionality of the feature vectors and the high runtime of the algorithm on the available hardware. The Dynamic Means algorithm, on the other hand, successfully identified the major flows of commercial aircraft across the United States along with their temporal patterns (the 12 most popular of which are shown in Figure 4-2) in orders of magnitude less time than both the DP-means and DDP mixture Gibbs sampling algorithms.

4.6 Summary

This chapter presented the Dynamic Means algorithm, a clustering algorithm for batch-sequential data containing temporally evolving clusters. This algorithm was derived from a low-variance asymptotic analysis of the Gibbs sampling algorithm for the dependent Dirichlet process mixture model. Synthetic and real data experiments demonstrated that the algorithm can provide a hard clustering with labelling accuracy comparable to similar algorithms, while posing a significantly lower computational

burden than those algorithms in practice. The speed of inference coupled with the convergence guarantees provided yield an algorithm which is suitable for use in time-critical applications, such as online model-based task allocation systems.

Chapter 5

SCORE: Simultaneous Clustering on Representation Expansion

5.1 Overview

In the preceding chapters, two new tools were developed: Markov decision process task allocation [100], and the Dynamic Means algorithm [106]. The first provides a mechanism for coordinating multiple autonomous agents under uncertainty, making the assumption that the stochastic model of the environment is fully known to the agents. The second provides a way to perform realtime, lifelong learning of such a model with the flexibility of Bayesian nonparametrics. Thus, the next, and final, portion of this thesis is dedicated to the development of a technology that bridges the gap between these two methods.

This chapter considers learning the model of a Markov decision process task that has an unknown number of realizations. For example, a target tracking task may involve a hostile, neutral, or evasive target, and this set of target types is unknown to the planning system a priori. The generalization of this problem, addressed by this chapter, is model learning in a Markov decision process (MDP) that exhibits an underlying multiple model structure. In particular, each observed episode from the MDP has a latent classification that determines from which of an unknown number of models it was generated, and the goal is to determine both the number and the pa-

rameterization of the underlying models. The main challenge in solving this problem arises from the coupling between the selection of a low-dimensional representation of the domain and the separation of observations into groupings. Present approaches to multiple model learning involve computationally expensive probabilistic inference over Bayesian nonparametric models. We propose Simultaneous Clustering on Representation Expansion (SCORE), an iterative scheme based on classical clustering and adaptive linear representations, which addresses this codependence in an efficient manner and guarantees convergence to a local optimum in model error. Empirical results on simulated domains demonstrate the advantages of SCORE when compared to contemporary techniques with respect to both sample and time complexity.

This chapter is based on the paper "Simultaneous Clustering on Representation Expansion for Learning Multimodel MDPs" [107], which was accepted in the 2013 Multidisciplinary Conference on Reinforcement Learning and Decision Making, written in collaboration with R. Klein, A. Geramifard, and J. How.

5.2 Introduction

A key component of model-based Reinforcement Learning (RL) techniques is to build an accurate model of a Markov Decision Process (MDP) from observed interactions [71, 72]. This chapter considers the problem of model learning in an MDP that exhibits an underlying multiple model structure. In particular, each of the unknown number of latent models contains a complete description of the transition dynamics and reward of the MDP, and each observed episode (or "trajectory") has a latent classification that determines from which model it was generated. An example of such a scenario is a pursuit task, where an agent seeks to capture a target whose movement model can be neutral (e.g. random walk) or defensive (e.g. avoiding the agent), and where the agent does not know about these behaviors a priori. Such missions are of great interest to the autonomous planning community: For example, in mobile robot path planning problems the models can describe various types of terrain, or for aircraft routing problems the models can describe the evolution of various

weather patterns.

Linear function approximation has scaled model-based RL techniques to larger problems using fixed bases [73, 76] and adaptive bases [97], yet these techniques have not incorporated the aforementioned decomposition of the underlying model. Because they do not exploit the decomposed structure, they often require a large number of samples to find an accurate model, and yield suboptimal policies that arise from averaging over all the underlying models. Multiple-model approaches [79, 80] explicitly incorporate the decomposition assumption into their modeling structure, yet they assume the number of model classes are known a priori, and assume a rigid model component structure that cannot grow over time like adaptive linear function approximation. Past work in Bayesian nonparametrics [68, 83] has addressed the problem of determining both the structure of the individual models and the number of models, but as of yet these methods are not practical in an RL setting as their inference procedures do not scale well to the large quantities of data typically observed in RL problems.

A successful model-based RL approach in multimodel problems must use a collection of observed trajectories to infer both the number of models and their parameterization, with which the agent can generate a collection of policies. There are two major challenges posed by this problem formulation. The first is that of *simultaneous model distinction and learning*: In order to learn the models, the algorithm must first separate the trajectories into groupings based on their latent classifications, but in order to separate the trajectories into such groupings, the algorithm must have a good parameterization of the models. The second is that *model distinction depends on the representation*: Since learning exact models is infeasible, a lower-dimensional representation is required, but the ability to separate trajectories into groupings depends on the chosen representation.

The main contribution of this chapter is the introduction of the simultaneous clustering on representation expansion (SCORE) architecture, to solve these two problems using a combination of clustering and linear representation adaptation. Given a fixed representation, the clustering algorithm is used both for the identification of model

classes and fitting the parameter of each individual class, while the representation adaptation algorithm ensures that the learned classes are not simply an artifact of a poor representation. Connections to the classical clustering literature lead to theoretical convergence guarantees. Finally, empirical results on a benchmark MDP scenario are presented, demonstrating the advantages of our proposed method compared to current state-of-the-art methods.

5.3 Simultaneous Clustering On Representation Expansion (SCORE)

5.3.1 Problem Formulation

The problem considered in the present work extends the traditional MRP learning scenario by allowing an unknown number of models to be responsible for generating the observations. More precisely, suppose there is a set $\mathcal{F} = \{f_k\}_{k=1}^{|\mathcal{F}|}$ of scalar fields of unknown cardinality $|\mathcal{F}|$ defined on $\mathcal{S}$, each associated with an MRP. Given a collection of N observed trajectories, each trajectory having been generated by one of the MRPs, the goal is to infer the set $\mathcal{F}$, i.e. both $|\mathcal{F}|$ and $f_k \; \forall \; k = 1, \ldots, |\mathcal{F}|$. As it is often intractable to learn the full representation of f for even just a single system, a linear feature representation of the system is used to reduce the problem to learning $\mathcal{F}_\Phi = \{f_{\Phi k}\}_{k=1}^{|\mathcal{F}_\Phi|}$. However, the capability to distinguish MRPs based on observed data is intimately linked to the particular chosen linear representation; thus, Φ itself must also be inferred in order to find the best grouping of the trajectories.

Based on the standard formulation of MRP model learning, the natural extension to the case with multiple models is a minimization of the sum of the squared predictive errors over all the trajectories from each MRP with respect to $f_{\Phi k}$. Let $z_i \in \{1, \ldots, |\mathcal{F}_\Phi|\}$ be the label denoting the index of the MRP from which episode y_i

was generated. Then the squared predictive error of trajectory i in MRP k is

$$\delta_y^2(i,k) = \sum_{j=1}^{T_i} (f_{ij} - f_{\Phi k}^T \phi_{ij})^2. \tag{5.1}$$

Finally, define $|\Phi|$ to be the number of features (columns) in Φ. Then, the overall goal of multimodel learning is to solve the optimization problem

$$\min_{\Phi, \mathcal{F}_\Phi, \{z_i\}_{i=1}^N} \lambda|\mathcal{F}_\Phi| + \eta|\Phi| + \sum_{k=1}^{|\mathcal{F}_\Phi|} \left[\nu \|f_{\Phi k}\|^2 + \sum_{i:z_i=k} \delta_y^2(i,k) \right], \tag{5.2}$$

where $\lambda|\mathcal{F}_\Phi|$ (with $\lambda > 0$) is a cost penalty on the complexity of the learned model, $\eta|\Phi|$ (with $\eta > 0$) is a cost penalty on the complexity of the representation, and $\nu\|f_{\Phi k}\|^2$ (with $\nu > 0$) is a regularization term. This problem formulation shares a strong connection to past single-model MRP learning techniques. If $|\mathcal{F}_\Phi| = 1$ is fixed, it reduces to model learning with an adaptive linear feature representation. If $|\Phi|$ is fixed as well, it reduces to a least squares model estimation problem. The introduction of the penalties $\lambda|\mathcal{F}_\Phi|$ and $\eta|\Phi|$ is motivated by the desire to keep both the number of MRPs and the dimensionality of the representation small; indeed, without these penalties, the solution to the optimization is found trivially by setting Φ to the tabular representation, and assigning each trajectory to its own MRP. The particular selection of a penalty proportional to $|\mathcal{F}_\Phi|$ is based upon past literature in the classical limits of Bayesian nonparametric clustering [59], while the penalty proportional to $|\Phi|$ arises from the implicit cost function minimized by many state-of-the-art representation adaptation algorithms, such as iFDD [103] and BEBFs [108], if they are modified to include an error reduction threshold for representation expansion. The optimization problem (5.2) is a mixed integer nonlinear program, with complexity that is nonlinear in the amount of data N for both exact and heuristic methods [109, 110] even given fixed $|\mathcal{F}_\Phi|$ and $|\Phi|$. Thus, finding an efficient heuristic method for this optimization problem is the focus of the present work.

A key insight that can be made about this problem formulation is that it shares a strong connection to the k-means problem [56]. In both the k-means problem and the

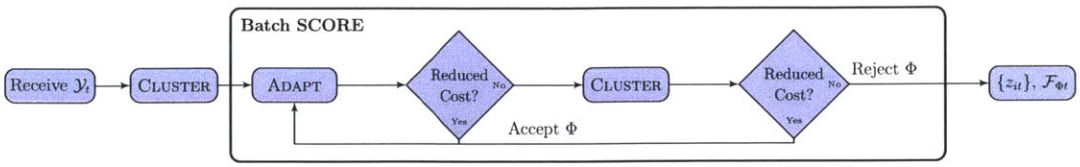


Figure 5-1: Algorithm block diagram.

optimization in equation (5.2), the goal is to simultaneously assign observations to clusters and to choose a parameter for each cluster that minimizes a sum of squared errors within that cluster. Drawing upon this insight an algorithmic architecture, simultaneous clustering on representation expansion (SCORE), that iterates between assigning trajectories to MRPs, finding the best parameter for each MRP, and adapting the linear representation is proposed. Then, the theoretical properties of SCORE are examined, and empirical results on a benchmark domain are presented.

5.3.2 Batch SCORE

The batch SCORE algorithm, shown in Figure 5-1, is an iterative algorithm for minimizing the objective (5.2). It starts by using the CLUSTER algorithm (5.3) to find an initial clustering of the data given an initial representation Φ. This yields a set of initial labels z_i $\forall i \in \{1, \ldots, N\}$ and models $f_{\Phi k}$ $\forall k \in \{1, \ldots, K\}$. Then, the ADAPT algorithm expands the representation using the predictive error for each observation with respect to its assigned cluster. If the expansion results in a decrease in (5.2), the expansion is accepted and the loop begins again. If it does not (due to the η penalty), the CLUSTER algorithm is run in an attempt to build a new clustering in the new representation. If that clustering yields an overall reduction in (5.2), the expansion is accepted and the loop begins again. If (5.2) is still not reduced, the algorithm terminates and returns $\mathcal{F}_\Phi$, Φ, and $\{z_i\}_{i=1}^N$. The mathematical details of the three major steps are as follows:

Cluster

$$f_{\Phi k} \leftarrow (\nu I + \sum_{i,j:z_i=k} \phi_{ij}\phi_{ij}^T)^{-1} \sum_{i,j:z_i=k} f_{ij}\phi_{ij} \tag{5.3}$$

$$z_i \leftarrow \underset{k}{\operatorname{argmin}} \begin{cases} \delta_y^2(i,k) & k \in \{1,\ldots,K\} \\ \lambda + \nu\|f_{\Phi k}\|^2 + \delta_y^2(i,k) & k = K+1 \end{cases} \tag{5.4}$$

Adapt

$$\theta^* \leftarrow \text{iFDD [72]} \tag{5.5}$$

$$\Phi \leftarrow \begin{bmatrix} \Phi & \theta^* \end{bmatrix} \tag{5.6}$$

The parameters of Batch SCORE are $\lambda, \eta, \nu \in \mathbb{R}_+$, where λ controls the addition of new clusters, η controls the addition of new features, and ν is the regularization parameter. K is defined as the number of clusters currently instantiated by the algorithm, and a new cluster is only created when the label update selects $K+1$ as the minimum cost index; thus, increasing λ reduces the number of clusters created. Note that $\delta_y^2(i, K+1)$ is equal to the distance from an observation y_i to the parameter found by solving for $f_{\Phi(K+1)}$ using only the data from that observation; this term is required to account for the fact that the best possible parameter for y_i has a nonzero error (unlike clustering problems in vector spaces). In particular, the $\delta_y^2(i, K+1)$ term compensates for representational distance (i.e. distance due to poor representation, rather than actual model discrepancy) and the varying length of trajectories (i.e. distance due to noise accumulating along longer trajectories), preventing the creation of unnecessary clusters. The regularization, ν, is present due to the possibility that the data within a single cluster does not visit sufficiently many unique states to make $\sum \phi_{ij}\phi_{ij}^T$ nonsingular. The per-iteration complexity of batch SCORE is dominated by the $O(D|\Phi|^2 + K|\Phi|^3)$ parameter update step in (5.3), where $D = \sum_{i=1}^{N} T_i$ is the total number of transitions observed.

The CLUSTER and ADAPT steps may be viewed as modified version of the DP-means [59] and iFDD [103] algorithms, respectively. The close relationship to these algorithms guarantees that Batch SCORE finds a local minimum in (5.2), and terminates in a finite number of iterations. These properties are demonstrated by Theorem 5.3.1 and Corollary 5.3.2:

Theorem 5.3.1 *Batch SCORE monotonically decreases the objective in (5.2).*

Proof It is sufficient to show that each individual update in (5.3)–(5.6) is guaranteed not to increase the cost. The model update in (5.3) is the solution of the minimization $f_{\Phi k} = \text{argmin}_{f_\Phi} \ \nu\|f_\Phi\|^2 + \sum_{i:z_i=k} \delta_y^2(i,k)$, so given any fixed Φ and $\{z_i\}_{i=1}^N$, (5.3) finds the lowest cost possible, and cannot increase (5.2). Likewise, the label update in (5.3) changes z_i from k to k': If $k' < K+1$ then $\delta_y^2(i,k') \leq \delta_y^2(i,k)$, and otherwise, $\lambda + \nu\|f_{\Phi K+1}\|^2 + \delta_y^2(i,K+1) \leq \delta_y^2(i,k)$; in either case the cost objective (5.2) decreases. Finally, the update (5.6) will not introduce a new feature (and incur a penalty of η) unless the sum of $\delta_y^2(i,k)$ across the dataset decreases by at least η; thus, the adaptation update decreases (5.2). As all of the individual updates monotonically decrease the cost objective, the theorem follows.

Corollary 5.3.2 *Batch SCORE finds a local optimum in (5.2) and terminates in a finite number of iterations.*

Proof The maximum number of linearly independent columns generated by applying the conjunction operator on the columns of a matrix is bounded. Given Theorem 5.3.1 and the finite number of possible data labelings the corollary follows. Note that while the algorithm may find multiple labelings with the same cost, it will not oscillate between them; it only makes a change to a label if the cost strictly decreases.

Remark Based on the proofs of Theorem 5.3.1 and Corollary 5.3.2, it can be seen that any combination of clustering algorithm and representation adaptation algorithm that are both guaranteed not to increase (5.2) and only introduce a finite number of new features will yield a convergent variant of Batch SCORE.

5.3.3 Incremental SCORE

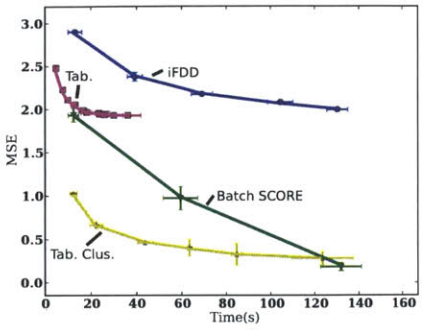
Building from the connection from Batch SCORE to the DP-means [59] algorithm, one can create an analogous incremental version of SCORE from Dynamic Means. Incremental SCORE can be used in place of batch SCORE once the dataset grows to a size that is computationally intractable to process in a single group. This algorithm is not discussed here for brevity, but is covered in the Appendix.

5.4 Experimental Results

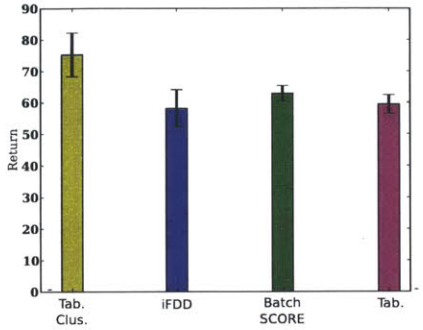
In this simulation experiment, we used SCORE to capture the effects of thermal updrafts on an unmanned aerial vehicle(UAV). The domain was a two-dimensional grid world with dimensions varied between 25×25, 40×40, and 50×50. The UAV could take actions selected from $\{\uparrow, \downarrow, \leftarrow, \rightarrow, \cdot\}$. Each state s had a mean updraft strength selected from $\mu_k(s) = \{2, 4, 6, 8, 10\}$, with $K = 3$ different latent mean updraft strength fields, and $k \in \{1, \ldots, K\}$. At the start of each episode, a label $z \sim \text{UNIFORM}(1, K)$ was sampled, and in each step t the UAV was given an altitude boost of $r_t \sim \mathcal{N}(\mu_z(s_t), \sigma = 0.25)$. The goal of the experiment was: 1) To learn the latent updraft fields $\mu_k(s)$ from a set of training data (collected with a uniform random policy); 2) To create a $Q(s, a)$ function for each learned field; and 3) To take actions by keeping track of the likelihood of each latent field given altitude boost observations, weighting the different $Q(s, a)$ functions accordingly, and acting greedily with respect to the weighted $Q(s, a)$ function (a Q_{MDP} approach [111]). We compared four approaches: SCORE (Batch SCORE), clustering with a tabular representation (Tab. Clus.), no clustering with iFDD (iFDD), and no clustering with a tabular representation (Tab.). The initial representation for the approaches with feature expansion consisted of an indicator function for each value of the two axes in the plane (e.g. 50 features for the 25×25 grid). The tabular representation had an indicator function feature for each grid cell (e.g. 625 features for the 25×25 grid).

The results are plotted in Figures 5-2a-5-2f. Figures 5-2a, 5-2c, and 5-2e show the true mean squared error (MSE) of altitude boost prediction and computational time (in seconds) required to converge (with 95 % confidence intervals after 30 trials). Each plotted point along each line represents an increment in the dataset size of 100 datapoints. Figures 5-2b, 5-2d, and 5-2f show the mean plan performance (with 95 % confidence intervals from 10 trials) using the model obtained after 1 minute of learning on the 25×25 domain, and after 10 minutes of learning on the 40×40 and 50×50 domains.

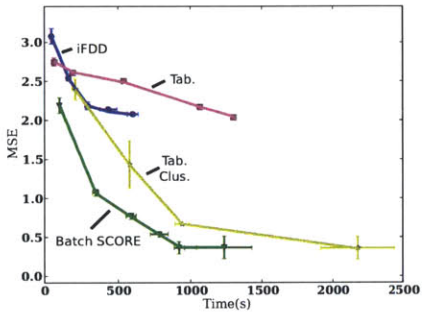
There are a number of points that can be made about these results. First and



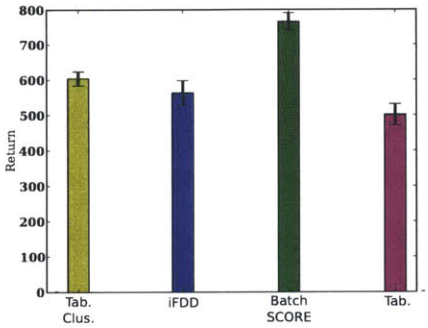
(a) 25×25



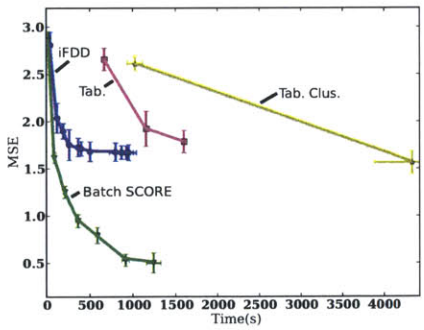
(b) 25×25



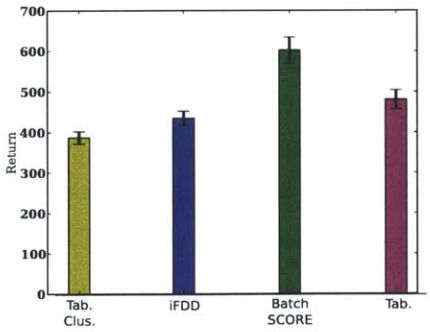
(c) 40×40



(d) 40×40



(e) 50×50



(f) 50×50

Figure 5-2: Model error (left column) and plan performance (right column) of batch SCORE (Batch SCORE), batch clustering with a tabular representation (Tab. Clus.), no clustering with a tabular representation (Tab.), and no clustering with feature expansion (iFDD). (5-2a-5-2b): 25×25 domain, (5-2c-5-2d): 40×40 domain, (5-2e-5-2f): 50×50 domain.

foremost, the approaches that do not account for the underlying multiple model structure (i.e. Tab./iFDD) fail to adequately predict altitude boost across the state-space (as evidenced by the high MSE at convergence), resulting in poor planning performance regardless of the problem size. Next, the naive approach that does consider the multiple model structure (using a tabular representation with clustering) does well in terms of both modeling and planning for small state-spaces, but as the size of the problem increases, its performance degrades rapidly. This degradation occurs both in terms of sample and time complexity, due to the rapidly increasing size of the representation. Based on the data for iFDD and Tabular without clustering, the computational complexity increase of Tab. Clus. is largely due to the clustering step with a high-dimensional representation (in particular, the $O(K|\Phi|^3)$ matrix inversion cost). SCORE, on the other hand, outperforms all of the other techniques with respect to the time and sample complexity of modeling. This results in a fast model-based reinforcement learning system in the presence of a problem with an underlying multiple-model structure.

5.5 Summary

This work addressed the problem of model learning in a Markov decision process (MDP) that exhibits an underlying multiple model structure using the Simultaneous Clustering on Representation Expansion (SCORE) algorithm. SCORE addresses the codependence between representation selection and observation clustering, and guarantees convergence to a local optimum in model error. Empirical results on a simulated benchmark domain demonstrated the advantage of this approach when compared to state-of-the-art tehcniques with respect to both sample and time complexity.

5.6 Appendix

While simple to implement and theoretically promising, the batch algorithm becomes computationally intractable as N grows. One can, however, limit the amount of data considered at a time by processing smaller batches of size $N_t \ll N$ in a sequence $t = 1, 2, \ldots$ (either by partitioning a fixed dataset, or by considering windows of size N_t in an online setting), reducing the per-iteration complexity of SCORE to $O(D_t|\Phi|^2 + K|\Phi|^3)$ where $D_t \ll D$. Doing so requires a modification to the optimization (5.2); the information about $\mathcal{F}_\Phi$ and Φ learned from one batch t must carry forward to the next batch at $t+1$, and so on. In other words, if a model $f_{\Phi kt}$ is present in batch t, and was most recently present in a batch Δt_k steps earlier, denoted $f_{\Phi k(t-\Delta t_k)}$, there must be a way of transferring the learned information about $f_{\Phi k(t-\Delta t_k)}$ to $f_{\Phi kt}$. Likewise, representational errors present in one batch with respect to Φ must be carried forward to the next batch in some way.

The Incremental SCORE algorithm accomplishes these goals with modified versions of the steps in (5.3) - (5.6). In this section, define $f'_{\Phi k} \equiv f_{\Phi k(t-\Delta t_k)}$, $\Phi' \equiv \Phi_{t-\Delta t_k}$, $\Phi \equiv \Phi_t$, and $f_{\Phi k} \equiv f_{\Phi kt}$ for notational brevity. When comparing $f_{\Phi k}$ with $f'_{\Phi k}$, $f'_{\Phi k}$ is the knowledge of past data projected onto Φ', and all knowledge of past data in the subspace orthogonal to Φ' has been lost. Therefore, the difference between the old and new parameters is only compared in the old Φ' subspace. This allows $f_{\Phi k}$ to vary freely to fit the data within the new portion of the Φ subspace, and provides a prior using the old parameter $f'_{\Phi k}$ within the old subspace. Thus, let the error between the old and new parameters be

$$\delta_M^2(k) = \|\Sigma^{\frac{1}{2}}\Phi'\left(f'_{\Phi k} - (\Phi'^T\Sigma\Phi')^{-1}\Phi'^T\Sigma\Phi f_{\Phi k}\right)\|_2^2$$

where Σ is a diagonal matrix with entries equal to the stationary distribution over the states in the Markov system (in practice, the stationary distribution is approximated sparsely using observed samples). Further, define $\mathcal{C}_t = \{f'_{\Phi k}, w_{k(t-\Delta t_k)}, \Delta t_k\}_{k=1}^{|\mathcal{C}_t|}$ to be the set of old parameters $f'_{\Phi k}$, their ages Δt_k and positive weights $w_{k(t-\Delta t_k)}$, and n_{kt} to be the sum over all transitions in all trajectories assigned to cluster k in batch t.

Finally, to transfer information about the errors with respect to Φ, the $\sum_{i,j}$ in (2.24) is taken over all transitions observed in *all* received batches of data. One can update the objective incrementally by adding to the numerator and denominator each time a new batch of data is received.

Given the above modifications, the following steps constitute Incremental SCORE, shown in the architecture in Figure 5-1:

Cluster

$$\gamma_{kt} \leftarrow \begin{cases} \left(\left(w_{k(t-\Delta t_k)}\right)^{-1} + \Delta t_k \tau\right)^{-1} & k \leq |\mathcal{C}_t| \\ 0 & k > |\mathcal{C}_t| \end{cases}$$

$$A \equiv \Phi^T \Sigma \Phi', \quad B \equiv \Phi'^T \Sigma \Phi' \tag{5.7}$$

$$f_{\Phi kt} \leftarrow (\gamma_{kt} A B^{-1} A^T + \nu I + \sum_{i,j:z_{it}=k} \phi_{ij} \phi_{ij}^T)^{-1} (\gamma_{kt} A f'_{\Phi k} + \sum_{i,j:z_{it}=k} f_{ij} \phi_{ij})$$

$$z_{it} \leftarrow \underset{k}{\operatorname{argmin}} \begin{cases} \delta_y^2(i,k) & n_{kt} > 0 \\ \nu \|f_{\Phi k}\|^2 + \delta_y^2(i,k) + \gamma_{kt} \delta_M^2(k) & n_{kt} = 0, k \leq |\mathcal{C}_t| \\ \lambda + \nu \|f_{\Phi k}\|^2 + \delta_y^2(i,k) & n_{kt} = 0, k > |\mathcal{C}_t| \end{cases} \tag{5.8}$$

Adapt

$$\theta^* \leftarrow \text{iFDD (see [72])} \tag{5.9}$$

$$\Phi \leftarrow \begin{bmatrix} \Phi & \theta^* \end{bmatrix} \tag{5.10}$$

Update $\mathcal{C}_t$

$$w_{kt} \leftarrow \gamma_{kt} + n_{kt}$$

$$\Delta t_k \leftarrow \begin{cases} 1 & n_{kt} > 0 \\ \Delta t_k + 1 & n_{kt} = 0 \end{cases} \tag{5.11}$$

The parameter $\tau > 0$ controls how quickly the weight of old information decays (increasing τ causes old information to decay faster). The algorithm differs from

Batch SCORE most noticeably in the CLUSTER step. The parameter γ_{kt} is set at the beginning of each CLUSTER step, and controls the weight on prior information in (5.7). During the label assignment step, the incremental algorithm can "revive" an old cluster; here, $\delta_M^2(k)$ is the error between $f_{\Phi kt}$ and $f_{\Phi k(t-\Delta t_k)}$ after using only observation y_{it} to compute the parameter update in (5.7). Once both CLUSTER and ADAPT terminate without making changes to the representation or clusters, $\mathcal{C}_t$ is updated, a new batch of data $\mathcal{Y}_t$ is received, and the loop repeats.

This algorithm may be seen as a modified version of Dynamic Means [106]. Due to the connection to this classical clustering method, incremental SCORE is guaranteed to converge in a finite number of iterations:

Theorem 5.6.1 *For each batch of data $\mathcal{Y}_t$, the inner clustering/representation expansion loop of Incremental SCORE monotonically decreases the following objective:*

$$\eta|\Phi| + \sum_{k:n_{kt}>0} \lambda\left[\Delta t_k = 0\right] + \gamma_{kt}\delta_M^2(k) + \nu\|f_{\Phi kt}\|^2 + \sum_{i:z_{it}=k} \delta_y^2(i,k) \tag{5.12}$$

Corollary 5.6.2 *For each batch of data $\mathcal{Y}_t$, Incremental SCORE finds a local optimum in (5.12) and terminates in a finite number of iterations.*

Remark The proofs of Theorem 5.6.1 and Corollary 5.6.2 follow the same logic as those of Theorem 5.3.1 and Corollary 5.3.2. Note that the objective in (5.12) is closely related to that in (5.2), as one reduces to the other when $\gamma_{kt} = 0$ and $\Delta t_k = 0$ for all clusters (i.e. when processing just one batch of data).

Chapter 6

Conclusions

6.1 Summary and Contributions

This thesis addressed the problem of tractable, fast multiagent learning and planning under uncertainty in a dynamic, evolving environment. Three core tools were developed.

The first was MDP task allocation, a polynomial-time algorithm for coordination between autonomous agents completing stochastic tasks modelled as Markov decision processes. This algorithm does not require particle sampling (in contrast to past approaches), and is scalable to situations involving a large number of agents and tasks, while capable of incorporating the generality of MDP models of those tasks. A complexity analysis provided a theoretical solution quality/computational workload tradeoff that will find use in the design of autonomous systems employing this approach. Empirical results for a problem of a size much larger than what is currently manageable by state-of-the-art approximate MDP planning techniques demonstrated the computational advantages of this approach, and verified the findings in the theoretical analysis.

The second was the Dynamic Means algorithm, a realtime clustering algorithm for batch-sequential data containing temporally evolving clusters. This algorithm was derived from a low-variance asymptotic analysis of the Gibbs sampling algorithm for the dependent Dirichlet process mixture model (DDPMM). A synthetic test on

moving Gaussian data, and a test on real ADS-B aircraft data across the United States of America, illustrated the low computational cost of the algorithm (in comparison to state-of-the-art Bayesian nonparametric inference techniques for the DDPMM), and the robustness of its label accuracy output to its parameter settings. The speed of inference using this algorithm, coupled with the theoretical analysis demonstrating the guaranteed convergence to a local cost optimum, yield an algorithm which is suitable for use in time-critical applications, such as the MDP task allocation system developed earlier.

The final contribution was Simultaneous Clustering on Representation Expansion (SCORE), a reinforcement learning paradigm that bridges the gap between the first two tools by combining classical clustering with feature expansion to discover the underlying multiple-model structure of Markov decision processes. SCORE addresses the codependence between representation selection and observation clustering, and guarantees convergence to a local optimum in model error. Empirical results on a simulated benchmark domain demonstrated the advantage of this approach when compared to nonclustering or nonexpanding methods, with respect to both sample and time complexity.

6.2 Future Work

Future work on MDP tak allocation includes: the formulation of a subclass of MDPs for which the value of a sequence of MDP tasks is submodular (enabling more efficient sequential greedy allocation algorithms that can prune candidate solutions); the investigation of how approximate MDP policies for tasks (e.g. those generated by trajectory-based planning methods) can be used within this framework to remove the assumption that policies are provided a priori; and the incorporation of machine learning techniques to learn a set of parameterized tasks using data collected from the environment. Future work on the Dynamic Means algorithm includes: devising a spectral relaxation based on the derived cost function, which will lead to an algorithm that is less sensitive to local optima and can be used to cluster arbitrary

graph data; the development of a smart probabilistic label initialization scheme that has theoretically guaranteed bounds on expected model error; and the creation of a decentralized version of the clustering algorithm based upon past decentralization of the DP-means algorithm. Finally, future work on SCORE should incorporate these changes to MDP task allocation and Dynamic Means, and methods for automatically tuning its various parameters should be investigated.

Bibliography

[1] L. Buşoniu, R. Babuška, and B. D. Schutter, "A comprehensive survey of multiagent reinforcement learning," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 38, no. 2, pp. 156–172, 2008.

[2] N. Roy, G. Gordon, and S. Thrun, "Planning under uncertainty for reliable health care robotics," *Field and Service Robotics*, vol. STAR 24, pp. 417–426, 2006.

[3] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics.* Cambridge, MA: MIT Press, 2005.

[4] B. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robotics and Autonomous Systems*, vol. 57, no. 5, pp. 469–483, 2009.

[5] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach, 2nd Edition.* Englewood Cliffs, NJ: Prentice-Hall, 2003.

[6] E. Alpaydin, *Introduction to Machine Learning (Adaptive Computation and Machine Learning).* The MIT Press, 2004.

[7] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics).* Springer, first ed., 2007.

[8] K. Korb and A. Nicholson, *Bayesian Artificial Intelligence.* Boca-Raton, FL: CRC Press, 2 ed., 2011.

[9] L. F. Bertuccelli and J. P. How, "Robust uav search for environments with imprecise probability maps," in *IEEE Conference on Decision and Control (CDC)*, (Seville, Spain), pp. 5680–5685, 12-15 December 2005.

[10] Y. W. Teh, *Dirichlet Process*, pp. 280–290. Encyclopedia of Machine Learning, Springer-Verlag New York Inc, 2011.

[11] Y. W. Teh, M. I. Jordan, M. J. Beal, and D. M. Blei, "Hierarchical dirichlet processes," *Journal of the American Statistical Association*, vol. 101, no. 476, pp. 1566–1581, 2006.

[12] D. Lin, E. Grimson, and J. Fisher, "Construction of dependent dirichlet processes based on poisson processes," in *Neural Information Processing Systems*, 2010.

[13] R. Thibaux and M. I. Jordan, "Hierarchical beta processes and the indian buffet process," in *International Conference on Artificial Intelligence and Statistics*, vol. 11, pp. 564–571, 2007.

[14] E. B. Fox, *Bayesian Nonparametric Learning of Complex Dynamical Phenomena.* PhD thesis, Massachusetts Institute of Technology, Cambridge MA, December 2009.

[15] C. Rasmussen and C. Williams, *Gaussian Processes for Machine Learning.* MIT Press, Cambridge, MA, 2006.

[16] R. Neal, "Markov chain sampling methods for dirichlet process mixture models," *Journal of computational and graphical statistics*, pp. 249–265, 2000.

[17] D. M. Blei and M. I. Jordan, "Variational inference for dirichlet process mixtures," *Bayesian Analysis*, vol. 1, no. 1, pp. 121–144, 2006.

[18] M. Hoffman, D. Blei, C. Wang, and J. Paisley, "Stochastic variational inference," *arXiv ePrint 1206.7051*, 2012.

[19] C. M. Carvalho, H. F. Lopes, N. G. Polson, and M. A. Taddy, "Particle learning for general mixtures," *Bayesian Analysis*, vol. 5, no. 4, pp. 709–740, 2010.

[20] D. J. White, "A survey of applications of markov decision processes," *Journal of the Operational Research Society*, vol. 44, no. 11, 1993.

[21] J. D. Redding, *Approximate Multi-Agent Planning in Dynamic and Uncertain Environments.* PhD thesis, Massachusetts Institute of Technology, Cambridge MA, February 2012.

[22] S. S. Ponda, *Robust Distributed Planning Strategies for Autonomous Multi-Agent Teams.* PhD thesis, Massachusetts Institute of Technology, Cambridge MA, September 2012.

[23] B. P. Gerkey and M. J. Mataric, "A formal analysis and taxonomy of task allocation in multi-robot systems," *International Journal of Robotics Research*, vol. 23(9), pp. 939–954, 2004.

[24] H.-L. Choi, L. Brunet, and J. P. How, "Consensus-based decentralized auctions for robust task allocation," *IEEE Transactions on Robotics*, vol. 25, pp. 912–926, August 2009.

[25] L. B. Johnson, H.-L. Choi, S. S. Ponda, and J. P. How, "Allowing non-submodular score functions in distributed task allocation," in *IEEE Conference on Decision and Control (CDC)*, Dec 2012 (to appear).

[26] L. B. Johnson, S. S. Ponda, H.-L. Choi, and J. P. How, "Asynchronous decentralized task allocation for dynamic environments," in *Proceedings of the AIAA Infotech@Aerospace Conference*, (St. Louis, MO), March 2011.

[27] U. Feige and J. Vondrak, "Approximation algorithms for allocation problems: Improving the factor of 1-1/e," in *IEEE Symposium on Foundations of computer Science*, pp. 667–676, 2006.

[28] A. Krause and C. Guestrin, "Near-optimal observation selection using submodular functions," in *In Proc. of Conf. on AI*, 2007.

[29] S. S. Ponda, L. B. Johnson, and J. P. How, "Distributed chance-constrained task allocation for autonomous multi-agent teams," in *American Control Conference (ACC)*, June 2012.

[30] S. S. Ponda, L. B. Johnson, A. Geramifard, and J. P. How, *Handbook of Unmanned Aerial Vehicles*, ch. Cooperative Mission Planning for Multi-UAV Teams. Springer, 2012 (to appear).

[31] J. Redding, A. Geramifard, A. Undurti, H. Choi, and J. How, "An intelligent cooperative control architecture," in *American Control Conference (ACC)*, (Baltimore, MD), pp. 57–62, July 2010.

[32] N. Meuleau, M. Hauskrecht, K.-E. Kim, L. Peshkin, L. P. Kaelbling, T. Dean, and C. Boutilier, "Solving very large weakly coupled markov decision processes," in *The Fifteenth National Conference on Artificial Intelligence*, (Madison, WI), pp. 165–172, 1998.

[33] C. Boutilier, R. I. Brafman, and C. Geib, "Prioritized goal decomposition of markov decision processes: Toward a synthesis of classical and decision theoretic planning," in *The Fifteenth Intl. Joint Conf. on AI (IJCAI)*, (Nagoya, Japan), pp. 1156–1162, 1997.

[34] S. Singh and D. Cohn, "How to dynamically merge markov decision processes," in *Advances in Neural Information Processing Systems*, (Cambridge), pp. 1057–1063, MIT Press, 1997.

[35] D. Dolgov and E. Durfee, "Computationally-efficient combinatorial auctions for resource allocation in weakly-coupled mdps," in *The Intl. Conf. on Autonomous Agents and Multi Agent Systems*, pp. 657–664, 2005.

[36] P. Plamondon, B. Chaib-draa, and A. R. Benaskeur, "A multiagent task associated mdp (mtamdp) approach to resource allocation," in *AAAI Spring Symposium on Distributed Plan and Schedule Management*, 2006.

[37] C. Boutilier, "Sequential optimality and coordination in multiagent systems," in *The Sixteenth Intl. Joint Conf. on AI (IJCAI)*, (Stockholm), pp. 478–485, 1999.

[38] S. Proper and P. Tadepalli, "Solving multiagent assignment markov decision processes," in *Proceedings of the 8th Intl. Joint Conf. on Autonomous Agents and Multiagent Systems*, pp. 681–688, 2009.

[39] C. Guestrin, M. Lagoudakis, and R. Parr, "Coordinated reinforcement learning," in *AAAI Symposium Series: Collaborative Learning Agents*, 2002.

[40] E. Fox, D. Choi, and A. Willsky, "Nonparametric bayesian methods for large scale multi-target tracking," in *Fortieth Asilomar Conference on Signals, Systems and Computers (ACSSC '06)*, pp. 2009 –2013, Nov 2006.

[41] E. Fox, E. B. Sudderth, and A. S. Willsky, "Hierachical Dirichlet processes for tracking maneuvering targets," in *Information Fusion, 2007 10th International Conference on*, July 2007.

[42] E. Fox, E. Sudderth, M. Jordan, and A. Willsky, "Nonparametric bayesian learning of switching linear dynamical systems," *Advances in Neural Information Processing Systems (NIPS)*, 2009.

[43] Y. W. Teh and M. Jordan, *Bayesian Nonparametrics in Practice*, ch. Hierarchical Bayesian Nonparametric Models with Applications. 2009.

[44] T. Ferguson, "A Bayesian Analysis of Some Nonparametric Problems," *The Annals of Statistics*, vol. 1, no. 2, pp. 209–230, 1973.

[45] C. E. Antoniak, "Mixtures of Dirichlet Processes With Applications to Bayesian Nonparametric Problems," *The Annals of Statistics*, vol. 2, no. 6, pp. 1152–1174, 1974.

[46] N. L. Hjort, "Nonparametric Bayes Estimators Based on Beta Processes in Models for Life History Data," *The Annals of Statistics*, vol. 18, no. 3, pp. 1259–1294, 1990.

[47] D. Steinberg, O. Pizarro, S. Williams, and M. Jakuba, "Dirichlet process mixture models for autonomous habitat classification," in *IEEE OCEANS Conference*, 2010.

[48] S. N. MacEachern, "Dependent nonparametric processes," in *Proceedings of the Bayesian Statistical Science Section*, American Statistical Association, 1999.

[49] Y. W. Teh, "Dirichlet processes," in *Encyclopedia of Machine Learning*, New York: Springer, 2010.

[50] R. M. Neal, "Markov chain sampling methods for dirichlet process mixture models," *Journal of Computational and Graphical Statistics*, vol. 9, no. 2, pp. 249–265, 2000.

[51] M. Pitt and N. Shephard, "Filtering via simulation: Auxiliary particle filters," *Journal of the American Statistical Association*, vol. 94, no. 446, pp. 590–599, 1999.

[52] F. Doshi-Velez and Z. Ghahramani, "Accelerated sampling for the indian buffet process," in *Proceedings of the International Conference on Machine Learning*, 2009.

[53] F. Endres, C. Plagemann, C. Stachniss, and W. Burgard, "Unsupervised discovery of object classes from range data using latent dirichlet allocation," in *Robotics Science and Systems*, 2005.

[54] M. Luber, K. Arras, C. Plagemann, and W. Burgard, "Classifying dynamic objects: An unsupervised learning approach," in *Robotics Science and Systems*, 2004.

[55] Z. Wang, M. Deisenroth, H. B. Amor, D. Vogt, B. Schölkopf, and J. Peters, "Probabilistic modeling of human movements for intention inference," in *Robotics Science and Systems*, 2008.

[56] S. P. Lloyd, "Least squares quantization in pcm," *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, 1982.

[57] D. Pelleg and A. Moore, "X-means: Extending k-means with efficient estimation of the number of clusters," in *Proceedings of the 17th International Conference on Machine Learning*, 2000.

[58] R. Tibshirani, G. Walther, and T. Hastie, "Estimating the number of clusters in a data set via the gap statistic," *Journal of the Royal Statistical Society B*, vol. 63, no. 2, pp. 411–423, 2001.

[59] B. Kulis and M. I. Jordan, "Revisiting k-means: New algorithms via bayesian nonparametrics," in *Proceedings of the 29th International Conference on Machine Learning (ICML)*, (Edinburgh, Scotland), 2012.

[60] T. Ishioka, "Extended k-means with an efficient estimation of the number of clusters," in *Proceedings of the 2nd International Conference on Intelligent Data Engineering and Automated Learning*, pp. 17–22, 2000.

[61] C. Hue, J.-P. L. Cadre, and P. Pérez, "Tracking multiple objects with particle filtering," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 38, no. 3, pp. 791–812, 2002.

[62] J. Vermaak, A. Doucet, and P. Pérez, "Maintaining multi-modality through mixture tracking," in *Proceedings of the 9th IEEE International Conference on Computer Vision*, 2003.

[63] J. Ko, D. J. Klein, D. Fox, and D. Hähnel, "Gp-ukf: Unscented kalman filters with gaussian process prediction and observation models," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1901–1907, 2007.

[64] A. Girard, C. E. Rasmussen, J. Quintero-Candela, and R. Murray-smith, "Gaussian process priors with uncertain inputs - application to multiple-step ahead time series forecasting," in *Advances in Neural Information Processing Systems*, pp. 529–536, MIT Press, 2003.

[65] Y. Engel, S. Mannor, and R. Meir, "Bayes meets Bellman: The Gaussian process approach to temporal difference learning.," in *International Conference on Machine Learning (ICML)* (T. Fawcett and N. Mishra, eds.), pp. 154–161, AAAI Press, 2003.

[66] C. Rasmussen and M. Kuss, "Gaussian processes in reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 16, pp. 751–759, 2004.

[67] G. S. Aoude, B. D. Luders, J. M. Joseph, N. Roy, and J. P. How, "Probabilistically safe motion planning to avoid dynamic obstacles with uncertain motion patterns," *Autonomous Robots*, vol. 35, no. 1, pp. 51–76, 2013.

[68] J. Joseph, F. Doshi-Velez, A. S. Huang, and N. Roy, "A bayesian nonparametric approach to modeling motion patterns," *Autonomous Robots*, vol. 31, no. 4, pp. 383–400, 2011.

[69] F. Doshi-Velez, "The infinite partially observable markov decision process," 2009.

[70] T. Campbell, S. S. Ponda, G. Chowdhary, and J. P. How, "Planning under uncertainty using nonparametric bayesian models," in *AIAA Guidance, Navigation, and Control Conference (GNC)*, August 2012.

[71] R. Sutton, "Integrated architectures for learning, planning, and reacting based on approximating dynamic programming," in *Proceedings of the 7th International Conference on Machine Learning*, pp. 216–224, 1990.

[72] A. Geramifard, T. J. Walsh, N. Roy, and J. How, "Batch iFDD: A Scalable Matching Pursuit Algorithm for Solving MDPs," in *Proceedings of the 29th Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, (Bellevue, Washington, USA), AUAI Press, 2013.

[73] R. Sutton, C. Szepesvári, A. Geramifard, and M. Bowling, "Dyna-style planning with linear function approximation and prioritized sweeping," in *Proceedings of the 25th International Conference on Machine Learning*, (Helsinki, Finland), 2008.

[74] H. Yao, R. S. Sutton, S. Bhatnagar, D. Dongcui, and C. Szepesvári, "Multi-step dynamic planning for policy evaluation and control," in *NIPS*, pp. 2187–2195, 2009.

[75] J. Asmuth, L. Li, M. Littman, A. Nouri, and D. Wingate, "A bayesian sampling approach to exploration in reinforcement learning," in *Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence (UAI)*, (Montreal, Canada), 2009.

[76] P. Poupart, N. Vlassis, J. Hoey, and K. Regan, "An Analytic Solution to Discrete Bayesian Reinforcement Learning," in *Proceedings of the 23rd International Conference on Machine Learning*, (Pittsburgh, PA), 2006.

[77] R. Brafman and M. Tennenholtz, "R-Max - A General Polynomial Time Algorithm for Near-Optimal Reinforcement Learning," *Journal of Machine Learning Research*, vol. 3, pp. 213–231, 2002.

[78] L. Li, M. Littman, and T. Walsh, "Knows what it knows: A framework for self-aware learning," in *Proceedings of the 25th International Conference on Machine Learning*, (Helsinki, Finland), 2008.

[79] M. Haruno, D. M. Wolpert, and M. Kawato, "MOSAIC Model for Sensorimotor Learning and Control," *Neural Computation*, vol. 13, no. 10, pp. 2201–2220, 2001.

[80] D. M. Wolpert and M. Kawato, "Multiple paired forward and inverse models for motor control," *Neural Networks*, vol. 11, pp. 1317–1329, 1998.

[81] K. Doya, K. Samejima, K. Katagiri, and M. Kawato, "Multiple model-based reinforcement learning," *Neural Computation*, vol. 14, pp. 1347–1369, 2002.

[82] N. Sugimoto, M. Haruno, K. Doya, and M. Kawato, "Mosaic for multiple-reward environments," *Neural Computation*, vol. 24, pp. 577–606, 2012.

[83] E. B. Fox, *Bayesian Nonparametric Learning of Complex Dynamical Phenomena*. PhD thesis, Massachusetts Institute of Technology, 2009.

[84] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *J. Mach. Learn. Res.*, vol. 3, pp. 993–1022, 2003.

[85] E. Fox, E. Sudderth, M. Jordan, and A. Willsky, "Bayesian nonparametric methods for learning markov switching processes," *IEEE Signal Processing Magazine*, vol. 27, no. 6, pp. 43 –54, 2010.

[86] P. Trautman and A. Krause, "Unfreezing the robot: Navigation in dense, interacting crowds," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2010.

[87] J. M. Joseph, "Nonparametric bayesian behavior modeling," Master's thesis, Massachusetts Institute of Technology, 2008.

[88] J. Sethuraman, "A constructive definition of dirichlet priors," *Statistia Sinica*, vol. 4, pp. 639–650, 1994.

[89] J. Boyd-Graber and D. M. Blei, "Syntactic topic models," in *Advances in Neural Information Processing Systems (NIPS)*, 2008.

[90] J. Kingman, *Poisson Processes.* Oxford University Press, 1993.

[91] A. Geramifard, J. Redding, N. Roy, and J. P. How, "UAV Cooperative Control with Stochastic Risk Models," in *American Control Conference (ACC)*, pp. 3393 – 3398, June 2011.

[92] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* NJ: John Wiley & Sons, Inc., 2005.

[93] L. Kocsis and C. Szepesvári, "Bandit based monte-carlo planning," in *European Conference on Machine Learning*, pp. 282–293, 2006.

[94] R. Parr, L. Li, G. Taylor, C. Painter-Wakefield, and M. L. Littman, "An analysis of linear models, linear value-function approximation, and feature selection for reinforcement learning," in *Proceedings of the 25th international conference on Machine learning*, ICML '08, (New York, NY, USA), pp. 752–759, ACM, 2008.

[95] A. Geramifard, *Practical Reinforcement Learning Using Representation Learning and Safe Exploration for Large Scale Markov Decision Processes.* PhD thesis, Massachusetts Institute of Technology, Department of Aeronautics and Astronautics, February 2012.

[96] A. Geramifard, N. K. Ure, S. Tellex, G. Chowdhary, N. Roy, and J. P. How, "A tutorial on linear function approximators for dynamic programming and

reinforcement learning," *Foundations and Trends in Machine Learning*, 2012 (submitted).

[97] N. K. Ure, A. Geramifard, G. Chowdhary, and J. P. How, "Adaptive Planning for Markov Decision Processes with Uncertain Transition Models via Incremental Feature Dependency Discovery," in *European Conference on Machine Learning (ECML)*, 2012.

[98] D. Bertsimas and R. Weismantel, *Optimization over integers*. Dynamic Ideas Belmont, MA, 2005.

[99] S. S. Ponda, L. B. Johnson, and J. P. How, "Risk allocation strategies for distributed chance-constrained task allocation," in *American Control Conference (ACC)*, June 2013 (to appear).

[100] T. Campbell, L. Johnson, and J. P. How, "Multiagent allocation of markov decision process tasks," in *American Control Conference (ACC)*, IEEE, 2013.

[101] J. G. Kemeny and J. L. Snell, *Finite Markov Chains*. New York: Springer-Verlag, 1976.

[102] M. L. Littman, T. L. Dean, and L. P. Kaelbling, "On the complexity of solving markov decision problems," in *The Eleventh Intl. Conf. on Uncertainty in AI (UAI)*, pp. 394–402, 1995.

[103] A. Geramifard, F. Doshi, J. Redding, N. Roy, and J. How, "Online discovery of feature dependencies," in *International Conference on Machine Learning (ICML)*, 2011.

[104] N. K. Ure, G. Chowdhary, J. Redding, T. Toksoz, J. How, M. Vavrina, and J. Vian, "Experimental demonstration of efficient multi-agent learning and planning for persistent missions in uncertain environments," in *AIAA Guidance, Navigation, and Control Conference (GNC)*, (Minneapolis, MN), August 2012.

[105] T. Campbell, L. Johnson, M. Liu, J. How, and L. Carin, "Multiagent allocation of bayesian nonparametric markov decision process tasks (poster)," in *NIPS Workshop on Bayesian Nonparametric Models for Reliable Planning and Decision-making Under Uncertainty*, 2012.

[106] T. Campbell, M. Liu, B. Kulis, and J. How, "Dynamic clustering via asymptotics of the dependent dirichlet process," in *Advances in Neural Information Processing Systems (NIPS)*, 2013 (submitted).

[107] T. Campbell, R. Klein, A. Geramifard, and J. How, "Simultaneous clustering on representation expansion for learning multimodel mdps," in *1st Multidisciplinary Conference on Reinforcement Learning and Decision Making*, 2013.

[108] R. Parr, C. Painter-Wakefield, L. Li, and M. Littman, "Analyzing feature generation for value-function approximation," in *Proceedings of the 24th International Conference on Machine Learning*, (Corvallis, OR), 2007.

[109] M. Inaba, N. Katoh, and H. Imai, "Applications of Weighted Voronoi Diagrams and Randomization to Variance-based k-Clustering," in *Proceedings of the 10th ACM Symposium on Computational Geometry*, pp. 332–339, 1994.

[110] D. Arthur, B. Manthey, and H. Röglin, "k-means has polynomial smoothed complexity," in *Proceedings of the 50th Symposium on Foundations of Computer Science*, 2009.

[111] M. Littman, A. Cassandra, and L. Kaelbling, "Learning policies for partially observable environments: scaling up," in *International Conference on Machine Learning (ICML)*, (San Francisco, CA), pp. 362–370, 1995.