COMBINING PRIOR KNOWLEDGE AND DATA: BEYOND THE BAYESIAN FRAMEWORK

BY

ARKADY EPSHTEYN

B.S., University of Illinois at Urbana-Champaign, 1996,
M.S., University of Illinois at Urbana-Champaign, 1998

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2006

Urbana, Illinois

Abstract

For many tasks such as text categorization and control of robotic systems, state-of-the art learning systems can produce results comparable in accuracy to those of human subjects. However, the amount of training data needed for such systems can be prohibitively large for many practical problems. A text categorization system, for example, may need to see many text postings manually tagged with their subjects before it learns to predict the subject of the next posting with high accuracy. A reinforcement learning (RL) system learning how to drive a car needs a lot of experimentation with the actual car before acquiring the optimal policy. An optimizing compiler targeting a certain platform has to construct, compile, and execute many versions of the same code with different optimization parameters to determine which optimizations work best. Such extensive sampling can be time-consuming, expensive (in terms of both expense of the human expertise needed to label data and wear and tear on the robotic equipment used for exploration in case of RL), and sometimes dangerous (e.g., an RL agent driving the car off the cliff to see if it survives the crash). The goal of this work is to reduce the amount of training data an agent needs in order to learn how to perform a task successfully. This is done by providing the system with prior knowledge about its domain. The knowledge is used to bias the agent towards useful solutions and limit the amount of training needed.

We explore this task in three contexts: classification (determining the subject of a newsgroup posting), control (learning to perform tasks such as driving a car up the mountain in simulation), and optimization (optimizing performance of linear algebra operations on different hardware platforms). For the text categorization problem, we introduce a novel algorithm which efficiently integrates prior knowledge into large margin classification. We show that prior knowledge simplifies the problem by reducing the size of the hypothesis space. We also provide formal convergence guarantees for our algorithm. For reinforcement learning, we introduce a novel framework for defining planning problems in terms of qualitative statements about the world (e.g., "the faster the car is going, the more likely it is to reach the top of the mountain"). We present an algorithm based on policy iteration for solving such qualitative problems and prove its convergence. We also present an alternative framework which allows the user to specify prior knowledge quantitatively in form of a Markov Decision Process (MDP). This prior is used to focus exploration on those regions of the

world in which the optimal policy is most sensitive to perturbations in transition probabilities and rewards. Finally, in the compiler optimization problem, the prior is based on an analytic model which determines good optimization parameters for a given platform. This model defines a Bayesian prior which, combined with empirical samples (obtained by measuring the performance of optimized code segments), determines the maximum-a-posteriori estimate of the optimization parameters.

Acknowledgments

I am grateful to my parents, Joseph and Alla, for providing support and encouragement which made this thesis work possible. I am also grateful to my adviser, Gerald DeJong, for his guidance and support, as well as his valuable ideas about the nature and applicability of prior knowledge in learning systems. I would like to thank my wife Helen for her patience and understanding. I am also indebted to my parents-in-law, Henry and Maria, for sharing with me the wisdom and experience they acquired while working on their own PhDs.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

E^3	Explicit Explore or Exploit Algorithm.
i.i.d.	independent identically distributed.
LDA	Linear Discriminant Analysis.
MDP	Markov Decision Process.
QMDP	Qualitative Markov Decision Process.
QRL	Qualitative Reinforcement Learning.
RL	Reinforcement Learning.
VC	Vapnik-Chervonenkis.
SOCP	Second Order Cone Programming.
SVM	Support Vector Machine.

Publication Note

Parts of this dissertation are based on the following published papers:

- Qualitative Reinforcement Learning
  with Gerald DeJong.
  International Conference on Machine Learning 2006.

- Generative Prior Knowledge for Discriminative Classification
  with Gerald DeJong.
  Journal of Artificial Intelligence Research, vol. 27.

- Rotational Prior Knowledge for SVMs
  with Gerald DeJong.
  European Conference on Machine Learning 2005.

- Analytic Models and Empirical Search: A Hybrid Approach to Code Optimization
  with Maria Garzaran and Gerald DeJong and David Padua and Gang Ren and Xiaoming Li and Kamen Yotov and Keshav Pingali,
  International Workshop on Languages and Compilers for Parallel Computing, 2005

Chapter 1

Introduction

1.1 Need for Prior Knowledge

Recent years have seen a tremendous amount of progress in the accuracy of classification and regression and tractability of reinforcement learning. However, these systems require a lot of training data to achieve acceptable performance. Consider, for example, a text categorization problem. This problem is often studied in the context of classifying newsgroup postings. Given a posting, the goal of the system is to determine which newsgroup the posting came from. While a state-of-the art classifier such as support vector machine (SVM) achieves excellent performance on this task, it needs to see a lot of postings labeled with their newsgroup names during the training stage. For many domains, acquiring such labeled postings is impractical. Intelligent user interfaces, for example, must adopt to the behavior of an individual after a limited amount of interaction to be useful. Medical systems diagnosing rare diseases have to generalize well after seeing very few examples. Natural language processing systems learning to identify infrequent social events (e.g., revolutions and wars) from news articles have access to very few training examples. Moreover, they rely on manually labeled data for training, and such data is expensive to obtain. Therefore, reducing the required amount of training data for a classification algorithm is an important problem. A great deal of effort has been concentrated in recent years on solving the problem of learning to classify from few labeled examples. Popular approaches include semisupervised learning (i.e., learning from both labeled and unlabeled data [12]), active learning (i.e., the classifier actively asks the user to label those examples which help it the most), and learning-to-learn (the classifier learns to share information between related learning tasks [72, 8, 35]).

In a typical reinforcement learning scenario, an agent obtains information about the world by actively exploring it. A car-driving agent, for example, explores the world by controlling the car and adjusting its behavior based on the rewards it receives. The agent may need a lot of exploration before it learns the optimal policy. In fact, reinforcement learning algorithms which come with theoretic guarantees (such as E^3 [48] and $R-MAX$ [15]) require the agent to aggressively explore its environment. However, such an

exploration strategy may place the agent in dangerous situations (e.g., it may crash a car multiple times before it learns to avoid this undesirable behavior). In reinforcement learning literature, the problem of avoiding dangerous situations during exploration has been studied recently by Abbeel et. al. [4]. The proposed solution is apprenticeship learning: the agent learns from training trajectories demonstrated by the domain expert. The problem of reducing the amount of exploration in reinforcement learning is also beginning to receive attention. An algorithm suggested in [5] uses a user-specified model of the environment (formulated as an MDP) as an approximate starting point for RL exploration. This model is corrected as the agent learns more about its environment.

The compiler optimization problem we address is that of determining high-level program transformation parameters such as array tiling for matrix multiplication (matrix multiplication is a key operation used by many linear algebra subroutines). This problem is platform-dependent (the optimal amount of tiling, for example, depends on the size of the cache) and must be tuned individually for each new hardware architecture. One popular approach to optimization [77] determines the correct optimization parameters via near-exhaustive sampling: programs with different tilings are generated, compiled, executed, and their performance is measured. The best-performing program determines the best tiling for a given platform. Speeding up the search makes it feasible to apply it to architectures with larger caches and as a benchmark for evaluating alternative machine design (i.e., a designer may use the optimization procedure to find out how fast linear algebra operations can be performed on a new machine).

In all of these domains, gathering information about the environment is an expensive, time-consuming, and sometimes dangerous process. In the rest of the thesis, I will describe how to decrease the reliance of agents on labeled training data by utilizing prior domain knowledge. A unifying characteristic of our domain knowledge is that it describes the way the world works, not how to solve a given problem. Arguably, this kind of knowledge is easier for domain experts to express because most of modern physical science is concerned with modeling how the world works. More importantly, this kind of knowledge generalizes to multiple learning problems (we demonstrate this with text categorization in Chapter 2). Even more importantly, this kind of knowledge already exists for many domains. In text categorization, for example, we make use of existing ontologies defining relations between words. In compiler optimization, we use an existing model which calculates the approximately optimal tiling for a given cache. Utilization of existing domain theories removes the need for expensive human expertise required to construct a domain theory for each new learning task.

1.2 Beyond the Bayesian Framework

Prior knowledge is traditionally handled within the Bayesian framework. This framework imposes a prior probability distribution on the space of possible worlds. This distribution could encode domain experts' commonsense intuitions about the world. For example, in the car driving world it is reasonable to make the world in which cars turn right when the steering wheel is turned right more probable a-priori than the world in which cars turn left. The agent uptdates this probability distribution as it gathers more information about the actual world. This update is done using standard rules of probability [62]. For example, an agent which experiments with a very unusual car wired to turn left when the steering wheel is turned right will eventually learn that its prior is incorrect as it gains more confidence about the car's behavior.

While intuitively appealing, the Bayesian framework suffers from the following serious shortcomings:

- It is not obvious how to solicit prior probabilities. Stating that a car will turn right when steered right is intuitive; stating that it will make a right turn with probability 90% and sharp right with probability 10% is not. Bayesian formulations of the reinforcement learning problem require quantitative statements about the car's behavior of the latter form.

- Some applications (such as reinforcement learning) require Bayesian experts to impose a prior on probability distributions. This forces domain experts to make statements such as: "The probability that the car will turn right with probability 90% is 80%". The meaning of such statements is extremely obscure.

- In most practical applications, making inferences is intractable for arbitrary priors. To ensure tractability, when the Bayesian framework is applied in practice, the prior probability distribution is represented in a simplified form (e.g., as a conjugate prior [62]), which no longer reflects experts' true beliefs and intuitions.

- Even such simplified representations usually cannot be processed efficiently, so only approximate inference is possible.

Thus, applying the Bayesian framework in practice is fraught with both computational and philosophical difficulties. In this work, we present several alternatives to the Bayesian framework. All of our algorithms bridge the gap between domain experts' desire to specify prior knowledge in form of intuitive statements and the need for the inferencing algorithm to interpret such intuitive descriptions quantitatively. In this thesis, we overscome the limitations of the Bayesian framework stated above by:

- presenting novel reinforcement learning algorithms which allow domain experts to specify their knowledge by stating how they believe the world behaves. This specification could be in form of definitive statements (e.g., "when the steering wheel is turned to the right, the car will turn right") or comparative statements (e.g., "when the steering wheel is turned right, the car is more likely to turn right than left").

- demonstrating how to interpret existing prior knowledge for novel tasks. In the context of text classification, we make use of existing ontologies (graphs linking related words to each other) to improve the accuracy of text classification.

- designing efficient novel algorithms for making inferences based on prior knowledge and experimental evidence.

Chapter 2

Prior Knowledge and Text Categorization

In this chapter, we will explore automatic integration of ontologies into classification tasks. An ontology defines how words are related to each other (synonyms, antonyms, etc.). A reasoning system which learns to predict the subject of a text snippet may learn from training data that the word "bullet" is highly correlated with text statements describing guns. However, it may never observe the word "rifle" in its training data, failing to realize that rifles are just as indicative as bullets for inferring the subject of a piece of text. An agent equipped with an ontology, on the other hand, would be able to combine the information learned from the training data (that bullets are important for prediction) with information in the ontology (that both rifles and bullets are related to guns) to improve the accuracy of its inferences. We present a system which learns how to use ontology for text classification. A novel aspect of the system is that the way in which the ontology was used is not hard-coded, but learned from the data.

While prior knowledge has proven useful for classification [65, 79, 38, 29, 69], it is notoriously hard to apply in practice because there is a mismatch between the form of prior knowledge that can be employed by classification algorithms (either prior probabilities or explicit constraints on the hypothesis space of the classifier) and the domain theories articulated by human experts. This is unfortunate because various ontologies and domain theories are available in abundance, but considerable amount of manual effort is required to incorporate existing prior knowledge into the native learning bias of the chosen algorithm. What would it take to apply an existing domain theory automatically to a classification task for which it was not specifically designed? The research described in this chapter has two important contributions: first, we present a novel framework for integrating prior knowledge into classification. Second, we demonstrate that it is feasible to use existing Ontologies such as WordNet [56] and Roget [55] to improve small-sample classification accuracy without any modifications. All of our classifiers make use of prior knowledge based on semantic distance between words in these ontologies - a straightforward and easily computable quantity. Our classification system exhibits robust performance with respect to these different ontologies constructed independently by different people. This is surprising because WordNet has a notoriously uneven structure (different parts of WordNet were constructed by different people), and Roget's structure is completely

different from WordNet: WordNet has rich semantic connections between words (synonyms, antonyms, meronyms, hypernyms, etc.) while Roget is a very shallow hierarchical ontology with hypernym relations only. However, our experiments show that semantic distance in both ontologies is sufficient for improving classification accuracy.

In Section 2.2, we evaluate schemes which enable the agent to learn how to use ontologies for classification. In each scheme, a classifier abstracts away from individual words and learns how to quantify the contribution of words semantically related to the label in the ontology to the classification confidence. Thus, the agent uses labeled examples not only to learn how to classify, but also how to interpret the ontology in the cotext of its classification task. We evaluate several classification algorithms in Section 2.2 and empirically demonstrate that the most effective way of interpreting an ontology is within the context of a generative LDA classifier. However, many successful learning algorithms (such as support vector machines) are discriminative. In Section 2.4, we present a novel algorithm which allows for the use of a generative prior in the discriminative classification setting.

Our algorithm assumes that the generative distribution of the data is given in the Bayesian framework: $Prob(data|model)$ and the prior $Prob'(model)$ are known. However, instead of performing Bayesian model averaging, we assume that a single model M^* has been selected a-priori, and the observed data is a manifestation of that model (i.e., it is drawn according to $Prob(data|M^*)$). The goal of the learning algorithm is to estimate M^*. This estimation is performed as a two-player sequential game of full information. The bottom (generative) player chooses the Bayes-optimal discriminator function $f(M)$ for the probability distribution $Prob(data|model = M)$ (*without taking the training data into account*) given the model M. The model M is chosen by the top (discriminative) player in such a way that its prior probability of occurring, given by $Prob'(M)$, is high, *and* it forces the bottom player to minimize the training-set error of its Bayes-optimal discriminator $f(M)$. This estimation procedure gives rise to a bilevel program. We show that, while the problem is known to be NP-hard, its approximation can be solved efficiently by iterative application of second-order cone programming.

Most of the work in this chapter appears in the paper [30]. Results in Section 2.2 have not been previously published.

2.1 Classification Preliminaries

To perform classification, each example x is represented by a vector of features. In the text classification problem, each posting $x = [x^1, .., x^n]$ is represented as a bag of words, with each binary-valued feature x^i

recording the presence of the corresponding word. In this work, we focus on the pairwise classification tasks. A generative classifier in this setting assumes that each [example, label] pair $[x \in \{0,1\}^n, y \in \{-1,1\}]$ is generated i.i.d. from a probability distribution $Prob(x|y)\Pi(y)$. Classification is done by a decision rule which assigns the most probable label to x: $y(x) = 1 \Leftrightarrow f(x) := Prob(x|1) - Prob(x|-1) > 0$. This decision rule is called Bayes optimal assuming that the labels are equiprobable a priori (i.e., $\Pi(y=-1) = \Pi(y=1) = \frac{1}{2}$). A discriminative classifier selects the decision rule $f(x)$ directly from some family of functions (e.g., $f(x;w) = w^T x$ parameterized by the weight vector w) to minimize the number of errors on the training data.

We consider the following classifiers in this chapter:

1. The simplest version of a Linear Discriminant Analysis (LDA) classifier posits that $x|(y=-1) \sim N(\mu_1, I)$ and $x|(y=1) \sim N(\mu_2, I)$ for posting x given label y, where $I \in \mathbb{R}^{(n\times n)}$ is the identity matrix and $N(\mu, \Sigma)$ stands for the Normal distribution with mean μ and covariance matrix Σ. Classification is done by a decision rule which assigns the most probable label to x. It is well-known (e.g. see [28]) that this decision rule is equivalent to the one given by the hyperplane $f(x) := (\mu_2 - \mu_1)^T x - \frac{1}{2}(\mu_2^T \mu_2 - \mu_1^T \mu_1) > 0$. The means $\widehat{\mu_i} = [\widehat{\mu_i^1}, .., \widehat{\mu_i^n}]$ are estimated via maximum likelihood from the training data $[x_1, y_1], .., [x_m, y_m]$[1].

2. The Naive Bayes classifier assumes that $Prob(x|y) = \prod_{j=1}^{n} p(x^j|y)$, with $\widehat{p(x^j|y)}$ estimated from the training data via maximum likelihood. Classification is done by assigning the most probable label to x.

3. The discriminative SVM classifier sets the separating hyperplane to directly minimize the number of errors on the training data:
$[\widehat{w} = [\widehat{w^1}, .., \widehat{w^n}], \widehat{b}] = \arg\min_{w,b} \|w\| \text{ s.t. } y_i(w^T x_i + b) \geq 1, i = 1, .., m.$

It is known that the decision rules $f(x)$ of all of these classifiers are hyperplanes in the n-dimensional space of features, i.e. they can all be represented in the form $f(x) = w^T x$, and the only difference is in how the weight vector w is chosen. Thus, comparing the performance of these classifiers is fair in a sense that only different methods for parameter selection are compared, while the hypothesis space remains the same.

2.2 Using Ontologies for classification

It seems intuitively obvious that an ontology should improve classification performance. For example, knowing that the word 'artillery' is important in identifying the subject of a posting as 'guns' and that 'artillery'

[1]The standard LDA classifier assumes that $x|(y=-1) \sim N(\mu_1, \Sigma)$ and $x|(y=-1) \sim N(\mu_2, \Sigma)$ and estimates the covariance matrix Σ as well as the means μ_1, μ_2 from the training data. In our experiments, we take $\Sigma = I$.

is connected to 'shooter' in the ontology would imply that the word 'shooter' carries the same amount of relevant information for classification as 'artillery', even if it was never explicitly observed in the training sample. The only part which needs to be defined is precisely how to calibrate the ontology for classification. In particular, a natural measure of relatedness of two words in the ontology is that of a semantic distance, i.e., the length in links between the words in the ontology represented by a network (i.e., words are represented by nodes, and relationships - synonym, hypernym, meronym, etc. - are links between nodes). We must somehow quantify the semantic distance in a way which makes it useful for classification. Before describing how to do that, we explain our training framework. We examine the use of ontologies in the learning-to-learn framework [72, 8, 35]. Specifically, a classifier is trained on a given *training classification task* (e.g., for classifying postings into the newsgroups 'atheism' and 'guns') and then tested on a different *test classification task* (e.g., for classifying postings for the newsgroups 'middle east' and 'auto'). The only way for a classifier to generalize in this scenario is to use the original sample to acquire information about the ontology, and then to exploit this information to help it label examples from the test sample. The classifier uses the training task to learn to quantify semantic distances. In doing so, it learns how to assign weights to feature words based on their semantic distances from the labels. It can then use this information to classify postings from the test classification task based on their distances to new test task labels.

To compare the effectiveness of three different classifiers for this purpose, we examined the following training procedures:

1. It is intuitive to interpret information embedded in WordNet as follows: if the title of the newsgroup is 'guns', then all the words with the same semantic distance to 'gun' (e.g., 'artillery', 'shooter', and 'ordnance' with the distance of two) provide a similar degree of classification information. To quantify this intuition for LDA classification, let $l_{i,train} = [l^1_{i,train}, .., l^j_{i,train}, .., l^n_{i,train}]$ be the vector of semantic distances in WordNet between each feature word j and the label of each training task newsgroup $i \in \{1, 2\}$. Define $\chi_i(v) \triangleq \frac{\sum_{j:l^j_{i,train}=v} \widehat{\mu^j_i}}{|j:l^j_{i,train}=v|}, i = 1, 2$, where $|\cdot|$ denotes cardinality of a set. χ_i compresses information in $\widehat{\mu}_i$ based on the assumption that words equidistant from the newsgroup label are equally likely to appear in a posting from that newsgroup. To test the performance of this compressed classifier on a new task with semantic distances given by $l_{i,test}$, the generative distributions are reconstructed via $\mu^j_i := \chi_i(l^j_{i,test})$. Notice that if the classifier is trained and tested on the same task, applying the function χ_i is equivalent to averaging the components of the means of the generative distribution corresponding to the equivalence classes of words equidistant from the label. If the classifier is tested on a different classification task, the reconstruction process reassigns the averages based on the semantic distances to the new labels.

2. In Naive Bayes classification, the compression function is $\chi_i''(v,u) \triangleq \frac{\sum_{j:l^j_{1,train}=v, l^j_{2,train}=u} \widehat{p(x^j|i)}}{|j:l^j_{1,train}=v, l^j_{2,train}=u|}$ and reconstruction is done via $p(x^j|i) := \chi_i''(l^j_{1,test}, l^j_{2,test})$. $\chi_i''(v,u)$ defines the probability that a word with semantic distance of u to the label of the first newsgroup and v to the label of the second newsgroup is generated given that the posting's label is i. Just like in LDA, this probability is computed by averaging the probabilities of generating words distances u, v away from the labels.

3. In the discriminative setting, a feature vector z is appended to x, where $z(u,v) = \alpha \sum_{j:l^j_{1,train}=v, l^j_{2,train}=u} x^j$. Vector z is the vector of counts of all the features in each equivalence class $[u,v]$ (i.e., u links away from the first label and v from the second label) in the input instance x. Thus, the output of the classifier on x is given by $w^T[xz] = w_x^T x + w_z^T z$ and can be subdivided into the contribution of the actual instance x and the contribution of the abstracted instance z which subdivides features of x into bins $z(u,v)$ based on semantic distances in the ontology. The ontology features z are scaled by α, a user-specified constant which determines a trade-off between using the abstract features z and the actual instance x for classification. The training is done by solving the program $[\widehat{w}, \widehat{b}] = \arg\min_{w,b} \|w\| \text{ s.t. } y_i(w_x^T x_i + w_z^T z_i + b) \geq 1, i = 1, .., m$. The abstract part of the weight vector w_z is common for the training task and the test task (on the test task, $z(u,v)$ is computed with respect to the new labels).

We evaluated the effectiveness of each of these schemes for the task of sharing information between related classification tasks through the ontology. Two datasets were used in the experiments: 20-newsgroup [11] which contains labeled postings from 20 different newsgroups, and WebKB [2] which contains web pages of university professors, students, and staff. The datasets were preprocessed by removing words which could not be interpreted as nouns by WordNet to ensure that only one part of WordNet's hierarchy was exercised. We verified that this preprocessing has virtually no effect on classification accuracy. The following training/test newsgroup setups were created:

1. Newsgroups 'atheism','guns','hockey','autos' were combined in all possible pairs, and all thirty possible setups with one pair of newsgroups used for the training task and a different pair used for the test task were constructed. These were evaluated with both WordNet and Roget ontologies.

2. In the same way, thirty setups were created from the categories 'course','faculty','student','staff' from WebKB dataset. The label 'professor' was used instead of 'faculty' since it better describes the content of corresponding webpages.

The results of the experiment appear in Figure 2.2. For each setup, the performance of the three classifiers on the training task is shown. Each classifier was trained on 100% of the training task data and no test task

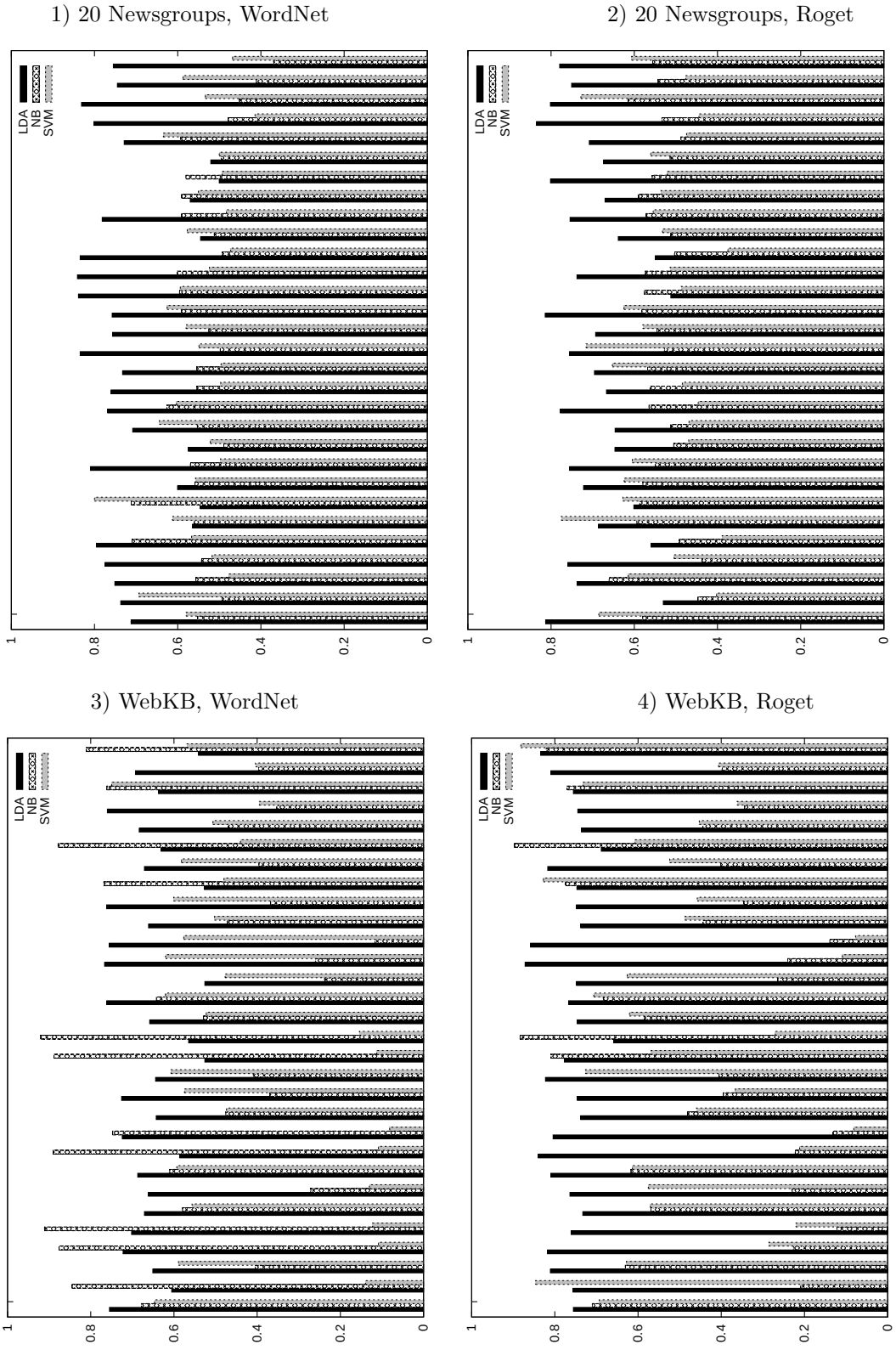


Figure 2.2.1: Using ontology for classification. Each set of three bars plots the accuracy of the three classifiers on one training/test task split. The accuracy is measured on the test task.

data. The accuracy on 100% of the training task data was measured. Only information shared through the use of the Ontology as described above was used to classify test task training instances. The LDA classifier outperforms both the Naive Bayes and the SVM classifier. In the next set of experiments, we compared the performance of all three classifiers after training them on 0.5% of test task data in addition to training task data. In the two-task setting, the classifiers are provided with two data sample $[x_{1t}, y_{1t}], .., [x_{m(t),t}, y_{m(t),t}]$ from the classification tasks $t \in \{\text{training task, test task}\}$.

1. For generative LDA classification, a Gaussian hyperprior is imposed on the parameters $\mu_i, i = 1, 2$ with confidence α. The maximum a-posteriori estimate $\widehat{\mu_i^*}$ is given by $\chi_i(l^j_{i,test}) * \frac{\alpha}{\alpha+n} + \widehat{\mu_{i,test}} * \frac{n}{\alpha+n}$, where $\widehat{\mu_{i,test}}$ is the maximum likelihood estimate of the mean of the Gaussian from the test task data.

2. For naive Bayes, a Beta hyperprior is imposed on the parameters $p(x^j|y)$ with confidence α to yield the maximum a posteriori estimates $\widehat{p^*(x^j|i)} = \chi_i''(l^j_{1,test}, l^j_{2,test}) * \frac{\alpha}{\alpha+n} + \widehat{p(x^j|i)} * \frac{n}{\alpha+n}$, where $\widehat{p(x^j|i)}$ is the maximum likelihood estimate of the mean of the conditional probability of word j given label i estimated from the test task data.

3. The discriminative SVM classifier solves the program $[\widehat{w}, \widehat{b}] = \arg\min_{w,b} \|w\| \text{ s.t. } y_{it}(w_x^T x_{it} + w_z^T z_{it} + b) \geq 1, i = 1, .., m(t), \forall t$.

This experiment enables us to compare the low-sample performance of the baseline classifier trained on 0.5% of the test task data only with that of the prior knowledge classifier. The results appear in Figure 2.2, which plots the performance of each classifier constrained with prior knowledge versus that of the corresponding baseline classifier. Points above the 45^o line outperform the baseline. Prior-constrained LDA and naive Bayes classification often improves performance and never decreases it significantly. On the other hand, in prior-constrained SVM classification, prior knowledge can hurt the classifier.

The conclusion is that the generative LDA classifier is superior to the discriminative SVM classifier in the task of sharing information via ontologies. However, our goal is not just to construct a classifier that performs well without seeing *any* (or seeing very few) examples of the test classification task. We also want a classifier which improves its behavior as it sees more labeled data from the test classification task. This presents us with a problem: one of the best-performing classifiers (and certainly the best on the text classification task according to the study by [47]). is SVM. Therefore, in the rest of this chapter, we focus on incorporating generative prior knowledge into the discriminative classification framework of support vector machines.

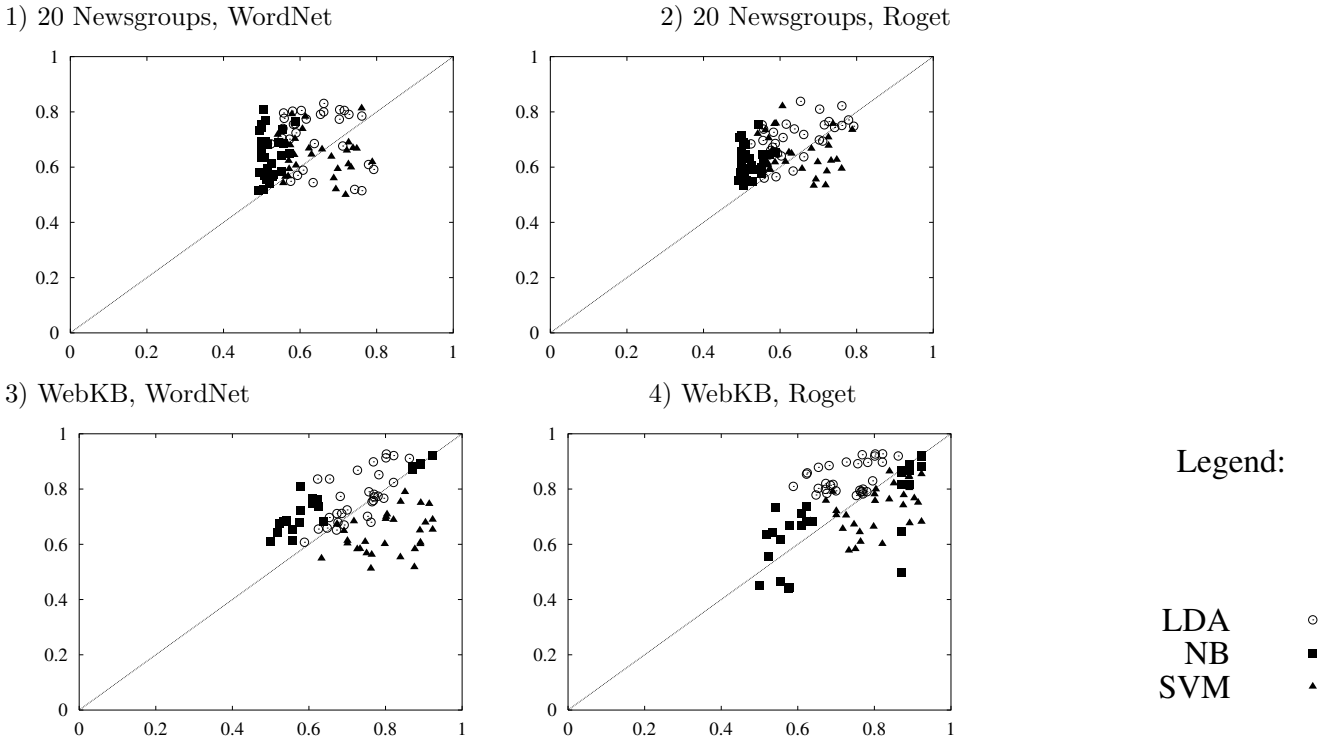


Figure 2.2.2: Comparing classifier's accuracy with and without prior knowledge. The x-coordinate of each point is the classifier's baseline accuracy on the test task, the y-coordinate is the accuracy of the prior-constrained classifier. The performance of three different classifiers on thirty tasks is plotted.

2.3 Second Order Cone Programming Preliminaries

It has been observed that constraints on the probability measure of a half-space can be captured by second-order cone constraints for Gaussian distributions (see, e.g., the tutorial by [51]). This allows for efficient processing of such constraints within the framework of second-order cone programming (SOCP). We intend to model prior knowledge with elliptical distributions, a family of probability distributions which generalizes Gaussians. In what follows, we give a brief overview of second-order cone programming and its relationship to constraints imposed on the Gaussian probability distribution. We also note that it is possible to extend the argument presented by Lobo et al. [51] to elliptical distributions.

Second-order cone program is a mathematical program of the form:

$$\min_x \; v^T x \tag{2.3.1}$$

$$\text{s.t.} \; \|A_i x + b_i\| \leq c_i^T x + d_i, \;\; i = 1, ..., N \tag{2.3.2}$$

where $x \in \mathbb{R}^n$ is the optimization variable and $v \in \mathbb{R}^n$, $A_i \in \mathbb{R}^{(k_i \text{x} n)}$, $b_i \in \mathbb{R}^{k_i}$, $c_i \in \mathbb{R}^n$, $d_i \in \mathbb{R}$ are problem

parameters ($\|\cdot\|$ represents the usual L_2-norm in this chapter). SOCPs can be solved efficiently with interior-point methods, as described by Lobo et al. [51] in a tutorial which contains an excellent overview of the theory and applications of SOCP.

We use the elliptical distribution to model distribution of the data a-priori. Elliptical distributions are distributions with ellipsoidally-shaped equiprobable contours. The density function of the n-variate elliptical distribution has the form $f_{\mu,\Sigma,g}(x) = c(\det\,\Sigma)^{-1}g((x-\mu)^T\Sigma^{-1}(x-\mu))$, where $x \in \mathbb{R}^n$ is the random variable, $\mu \in \mathbb{R}^n$ is the location parameter, $\Sigma \in \mathbb{R}^{(n\text{x}n)}$ is a positive definite $(n \times n)$-matrix representing the scale parameter, function $g(\cdot)$ is the density generator, and c is the normalizing constant. We will use the notation $X \sim E(\mu, \Sigma, g)$ to denote that the random variable X has an elliptical distribution with parameters μ, Σ, g. Choosing appropriate density generator functions g, the Gaussian distribution, the Student-t distribution, the Cauchy distribution, the Laplace distribution, and the logistic distribution can be seen as special cases of the elliptical distribution. Using an elliptical distribution relaxes the restrictive assumptions the user has to make when imposing a Gaussian prior, while keeping many desirable properties of Gaussians, such as:

1. If $X \sim E(\mu, \Sigma, g)$, $A \in \mathbb{R}^{(k\times n)}$, and $B \in \mathbb{R}^k$, then $AX + B \sim E(A\mu + B, A\Sigma A^T, g)$
2. If $X \sim E(\mu, \Sigma, g)$, then $E(X) = \mu$.
3. If $X \sim E(\mu, \Sigma, g)$, then $Var(X) = \alpha_g\Sigma$, where α_g is a constant that depends on the density generator g.

The following proposition shows that for elliptical distributions, the constraint $P(w^Tx+b \geq 0) \leq \eta$ (i.e., the probability that X takes values in the half-space $\{w^Tx+b \geq 0\}$ is less than η) is equivalent to a second-order cone constraint for $\eta \leq \frac{1}{2}$:

Proposition 2.3.1. *If $X \sim E(\mu, \Sigma, g)$, $Prob(w^Tx + b \geq 0) \leq \eta \leq \frac{1}{2}$ is equivalent to $-(w^T\mu + b)/\beta_{g,\eta} \geq \left\|\Sigma^{1/2}w\right\|$, where $\beta_{g,\eta}$ is a constant which only depends on g and η.*

Proof. The proof is identical to the one given by Lobo [51] and Lanckriet et al. [49] for Gaussian distributions and is provided here for completeness:

$$\text{Assume } Prob(w^Tx + b \geq 0) \leq \eta. \tag{2.3.3}$$

Let $u = w^Tx + b$. Let $\overline{u}$ denote the mean of u, and σ denote its variance. Then the constraint 2.3.3 can be written as

$$Prob(\frac{u - \overline{u}}{\sqrt{\sigma}} \geq -\frac{\overline{u}}{\sqrt{\sigma}}) \leq \eta. \tag{2.3.4}$$

By the properties of elliptical distributions, $\overline{u} = w^T\mu + b$, $\sigma = \sqrt{\alpha_g}\left\|\Sigma^{1/2}w\right\|$, and $\frac{u-\overline{u}}{\sqrt{\sigma}} \sim E(0,1,g)$. Thus, statement 2.3.4 above can be expressed as $Prob_{X\sim E(0,1,g)}(X \geq -\frac{w^T\mu+b}{\sqrt{\alpha_g}\left\|\Sigma^{1/2}w\right\|}) \leq \eta$, which is equivalent to $-\frac{w^T\mu+b}{\sqrt{\alpha_g}\left\|\Sigma^{1/2}w\right\|} \geq \Phi^{-1}(\eta)$, where $\Phi(z) = Prob_{X\sim E(0,1,g)}(X \geq z)$. The proposition follows with $\beta_{g,\eta} = \sqrt{\alpha_g}\Phi^{-1}(\eta)$. □

Proposition 2.3.2. *For any monotonically decreasing g, $Prob_{X\sim E(\mu,\Sigma,g)}(x) \geq \delta$ is equivalent to $\left\|\Sigma^{-1/2}(x-\mu)\right\| \leq \varphi_{g,c,\Sigma}$, where $\varphi_{g,c,\Sigma,\delta} = g^{-1}(\frac{\delta|\Sigma|}{c})$ is a constant which only depends on g, c, Σ, δ.*

Proof. Follows directly from the definition of $Prob_{X\sim E(\mu,\Sigma,g)}(x)$. □

2.4 Generative Prior via Bilevel Programming

We deal with the binary classification task: the classifier is a function $f(x)$ which maps instances $x \in \mathbb{R}^n$ to labels $y \in \{-1, 1\}$. In the generative setting, the probability densities $Prob(x|y = -1; \mu_1)$ and $Prob(x|y = 1; \mu_2)$ parameterized by $\mu = [\mu_1, \mu_2]$ are provided (or estimated from the data), along with the prior probabilities on class labels $\Pi(y = -1)$ and $\Pi(y = 1)$, and the Bayes optimal decision rule is given by the classifier

$$f(x|\mu) = sign(Prob(x|y = -1; \mu_1)\Pi(y = -1) - Prob(x|y = 1; \mu_2)\Pi(y = 1)),$$

where $sign(x) := 1$ if $x \geq 0$ and -1 otherwise. In LDA, for instance, the parameters μ_1 and μ_2 are the means of the two Gaussian distributions generating the data given each label.

Informally, our approach to incorporating prior knowledge is straightforward: we assume a two-level hierarchical generative probability distribution model. The low-level probability distribution of the data given the label $Prob(x|y; \mu)$ is parameterized by μ, which, in turn, has a known probability distribution $Prob'(\mu)$. The goal of the classifier is to estimate the values of the parameter vector μ from the training set of labeled points $[x_1, y_1]...[x_m, y_m]$. This estimation is performed as a two-player sequential game of full information. The bottom (generative) player, given μ, selects the Bayes optimal decision rule $f(x|\mu)$. The top (discriminative) player selects the value of μ which has a high probability of occurring (according to $Prob'(\mu)$) *and* which will force the bottom player to select the decision rule which minimizes the discriminative error on the training set. We now give a more formal specification of this training problem and formulate it as a bilevel program. Some of the assumptions are subsequently relaxed to enforce both tractability and flexibility.

We use an elliptical distribution $E(\mu_1, \Sigma_1, g)$ to model $X|y = -1$, and another elliptical distribution $E(\mu_2, \Sigma_2, g)$ to model $X|y = 1$. If the parameters $\mu_i, \Sigma_i, i = 1, 2$ are known, the Bayes optimal decision rule *restricted to the class of linear classifiers*[2] of the form $f_{w,b}(x) = sign(w^T x + b)$ is given by $f(x)$ which minimizes the probability of error among all linear discriminants: $Prob(error) = Prob(w^T x + b \geq 0|y = 1)\Pi(y = 1) + Prob(w^T x + b \leq 0|y = -1)\Pi(y = -1) = \frac{1}{2}(Prob_{X\sim E(\mu_1,\Sigma_1,g)}(w^T x + b \geq 0) + Prob_{X\sim E(\mu_2,\Sigma_2,g)}(w^T x + b \leq 0))$, assuming equal prior probabilities for both classes. We now model the uncertainty in the means of the elliptical distributions $\mu_i, i = 1, 2$ by imposing elliptical prior distributions on the locations of the means: $\mu_i \sim E(t_i, \Omega_i, g), i = 1, 2$. In addition, to ensure the optimization problem is well-defined, we maximize the margin of the hyperplane subject to the imposed generative probability constraints:

$$\min_{\mu_1,\mu_2} \|w\| \tag{2.4.1}$$

$$\text{s.t.} y_i(w^T x_i + b) \geq 1,\ i = 1, .., m \tag{2.4.2}$$

$$Prob_{\mu_i \sim E(t_i,\Omega_i,g)}(\mu_i) \geq \delta,\ i = 1, 2 \tag{2.4.3}$$

$$[w, b] \text{ solves } \min_{w,b}[Prob_{X\sim E(\mu_1,\Sigma_1,g)}(w^T x + b \geq 0) + Prob_{X\sim E(\mu_2,\Sigma_2,g)}(w^T x + b \leq 0)] \tag{2.4.4}$$

This is a bilevel mathematical program (i.e., an optimization problem in which the constraint region is implicitly defined by another optimization problem), which is strongly NP-hard even when all the constraints and both objectives are linear [43]. However, we show that it is possible to solve a reasonable approximation of this problem efficiently with several iterations of second-order cone programming. First, we relax the second-level minimization (2.4.4) by breaking it up into two constraints: $Prob_{X\sim E(\mu_1,\Sigma_1,g)}(w^T x + b \geq 0) \leq \eta$ and $Prob_{X\sim E(\mu_2,\Sigma_2,g)}(w^T x + b \leq 0) \leq \eta$. Thus, instead of looking for the Bayes optimal decision boundary, the algorithm looks for a decision boundary with low probability of error, where low error is quantified by the choice of η.

Propositions 2.3.1 and 2.3.2 enable us to rewrite the optimization problem resulting from this relaxation

[2] A decision rule *restricted to some class of classifiers H* is optimal if its probability of error is no larger than that of any other classifier *in H* [74].

as follows :

$$\min_{\mu_1,\mu_2,w,b} \|w\| \tag{2.4.5}$$

$$\text{s.t.} y_i(w^T x_i + b) \geq 1,\ i = 1, .., m \tag{2.4.6}$$

$$Prob_{\mu_i \sim E(t_i,\Omega_i,g)}(\mu_i) \geq \delta,\ i = 1, 2 \ \Leftrightarrow\ \left\|\Omega_i^{-1/2}(\mu_i - t_i)\right\| \leq \varphi, i = 1, 2 \tag{2.4.7}$$

$$Prob_{X \sim E(\mu_1,\Sigma_1,g)}(w^T x + b \geq 0) \leq \eta \ \Leftrightarrow\ -\frac{w^T\mu_1 + b}{\left\|\Sigma_1^{1/2} w\right\|} \geq \beta \tag{2.4.8}$$

$$Prob_{X \sim E(\mu_2,\Sigma_2,g)}(w^T x + b \leq 0) \leq \eta \ \Leftrightarrow\ \frac{w^T\mu_2 + b}{\left\|\Sigma_2^{1/2} w\right\|} \geq \beta \tag{2.4.9}$$

Notice that the form of this program does not depend on the generator function g of the elliptical distribution - only constants β and φ depend on it. φ defines how far the system is willing to deviate from the prior in its choice of a generative model, and β bounds the tail probabilities of error (Type I and Type II) which the system will tolerate assuming its chosen generative model is correct. These constants depend both on the specific generator g and the amount of error the user is willing to tolerate. In our experiments, we select the values of these constants to optimize performance. Unless the user wants to control the probability bounds through these constants, it is sufficient to assume a-priori only that probability distributions (both prior and hyper-prior) are elliptical, without making any further commitments.

Our algorithm solves the above problem by repeating the following two steps:

1. Fix the top-level optimization parameters μ_1 and μ_2. This step combines the objectives of maximizing the margin of the classifier on the training data and ensuring that the decision boundary is (approximately) Bayes optimal with respect to the given generative probability densities specified by the μ_1, μ_2.

2. Fix the bottom-level optimization parameters w, b. Expand the feasible region of the program in step 1 as a function of μ_1, μ_2. This step fixes the decision boundary and pushes the means of the generative distribution as far away from the boundary as the constraint (2.4.7) will allow.

The steps are repeated until convergence (in practice, convergence is detected when the optimization parameters do not change appreciably from one iteration to the next). Each step of the algorithm can be formulated as a second-order cone program:

Step 1. Fix μ_1 and μ_2. Removing unnecessary constraints from the mathematical program above and

pushing the objective into constraints, we get the following SOCP:

$$\min_{w,b} \rho \tag{2.4.10}$$

$$\text{s.t.} \rho \geq \|w\| \tag{2.4.11}$$

$$y_i(w^T x_i + b) \geq 1,\ i = 1, .., m \tag{2.4.12}$$

$$-\frac{w^T \mu_1 + b}{\left\|\Sigma_1^{1/2} w\right\|} \geq \beta \tag{2.4.13}$$

$$\frac{w^T \mu_2 + b}{\left\|\Sigma_2^{1/2} w\right\|} \geq \beta \tag{2.4.14}$$

Step 2. Fix w, b and expand the span of the feasible region, as measured by $\frac{w^T \mu_2 + b}{\left\|\Sigma_2^{1/2} w\right\|} - \frac{w^T \mu_1 + b}{\left\|\Sigma_1^{1/2} w\right\|}$. Removing unnecessary constraints, we get:

$$\max_{\mu_1, \mu_2} \frac{w^T \mu_2 + b}{\left\|\Sigma_2^{1/2} w\right\|} - \frac{w^T \mu_1 + b}{\left\|\Sigma_1^{1/2} w\right\|} \tag{2.4.15}$$

$$\text{s.t.} \left\|\Omega_i^{-1/2}(\mu_i - t_i)\right\| \leq \varphi,\ i = 1, 2 \tag{2.4.16}$$

The behavior of the algorithm is illustrated in Figure 2.4.1.

The following theorems state that the algorithm converges.

Theorem 2.4.1. *Suppose that the algorithm produces a sequence of iterates* $\left\{\mu_1^{(t)}, \mu_2^{(t)}, w^{(t)}, b^{(t)}\right\}_{t=0}^{\infty}$*, and the quality of each iterate is evaluated by its margin* $\left\|w^{(t)}\right\|$*. This evaluation function converges.*

Proof. Let $\mu_1^{(t)}, \mu_2^{(t)}$ be the values of the prior location parameters, and $w_1^{(t)}, b_1^{(t)}$ be the minimum error hyperplane the algorithm finds at the end of the t-th step. At the end of the $(t+1)$-st step, $w_1^{(t+1)}, b_1^{(t+1)}$ is still in the feasible region of the t-th step SOCP. This is true because the function $f(\frac{(w^{(t)})^T \mu_2 + b^{(t)}}{\left\|\Sigma_2^{1/2} w^{(t)}\right\|}, -\frac{(w^{(t)})^T \mu_1 + b^{(t)}}{\left\|\Sigma_1^{1/2} w^{(t)}\right\|}) = \frac{(w^{(t)})^T \mu_2 + b^{(t)}}{\left\|\Sigma_2^{1/2} w^{(t)}\right\|} - \frac{(w^{(t)})^T \mu_1 + b^{(t)}}{\left\|\Sigma_1^{1/2} w^{(t)}\right\|}$ is monotonically increasing in each one of its arguments when the other argument is fixed, and fixing μ_1 (or μ_2) fixes exactly one argument. If the solution $\mu_1^{(t+1)}, \mu_2^{(t+1)}$ at the end of the $(t+1)$-st step were such that $\frac{(w^{(t)})^T \mu_2^{(t+1)} + b^{(t)}}{\left\|\Sigma_2^{1/2} w^{(t)}\right\|} < \beta$, then f could be increased by fixing $\mu_1^{(t+1)}$ and using the value of $\mu_2^{(t)}$ from the beginning of the step which ensures that $\frac{(w^{(t)})^T \mu_2^{(t)} + b^{(t)}}{\left\|\Sigma_2^{1/2} w^{(t)}\right\|} \geq \beta$, which contradicts the observation that f is maximized at the end of the second step. The same contradiction is reached if $-\frac{(w^{(t)})^T \mu_1^{(t+1)} + b^{(t)}}{\left\|\Sigma_1^{1/2} w^{(t)}\right\|} < \beta$. Since the minimum error hyperplane from the previous iteration is in the feasible region at the start of the next iteration, the objective $\left\|w^{(t)}\right\|$ must decrease monotonically from one iteration

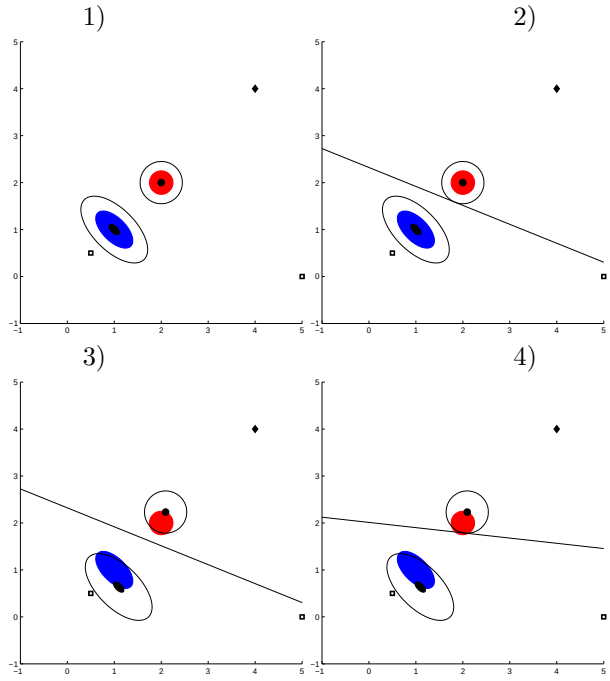


Figure 2.4.1: Steps of the iterative (hard-margin) SOCP procedure:
(The region where the hyperprior probability is larger than δ is shaded for each prior distribution. The covariance matrices are represented by equiprobable elliptical contours. In this example, the covariance matrices of the hyperprior and the prior distributions are multiples of each other. Data points from two different classes are represented by diamonds and squares.)

1. Data, prior, and hyperprior before the algorithm is executed.
2. Hyperplane discriminator at the end of step 1, iteration 1
3. Priors at the end of step 2, iteration 1
4. Hyperplane discriminator at the end of step 2, iteration 2

The algorithm converges at the end of step 2 for this problem (step 3 does not move the hyperplane).

to the next. Since it is bounded below by zero, the algorithm converges. □

In addition to the convergence of the objective function, the accumulation points of the sequence of iterates can be characterized by the following theorem:

Theorem 2.4.2. *The accumulation points of the sequence* $\left\{\mu_1^{(t)}, \mu_2^{(t)}, w^{(t)}, b^{(t)}\right\}$ *(i.e., limiting points of its convergent subsequences) have no feasible descent directions for the original optimization problem given by (2.4.5)-(2.4.9).*

Proof. See Appendix A.1. □

If a point has no feasible descent directions, then any sufficiently small step along any directional vector

will either increase the objective function, leave it unchanged, or take the algorithm outside of the feasible region. The set of points with no feasible descent directions is a subset of the set of local minima. Hence, convergence to such a point is a somewhat weaker result than convergence to a local minimum.

In practice, we observed rapid convergence usually within 2-4 iterations.

Finally, we may want to relax the strict assumptions of the correctness of the prior/linear separability of the data by introducing slack variables into the optimization problem above. This results in the following program:

$$\min_{\mu_1,\mu_2,w,b,\xi_i,\zeta_1,\zeta_2,\nu_1,\nu_2} \|w\| + C_1 \sum_{i=1}^{m} \xi_i + C_2(\zeta_1+\zeta_2) + C_3(\nu_1+\nu_2) \tag{2.4.17}$$

$$\text{s.t.} y_i(w^T x_i + b) \geq 1 - \xi_i,\ i = 1,..,m \tag{2.4.18}$$

$$\left\| \Omega_i^{-1/2}(\mu_i - t_i) \right\| \leq \varphi + \nu_i, i = 1,2 \tag{2.4.19}$$

$$-\frac{w^T\mu_1 + b}{\beta} \geq \left\| \Sigma_1^{1/2} w \right\| - \zeta_1 \tag{2.4.20}$$

$$\frac{w^T\mu_2 + b}{\beta} \geq \left\| \Sigma_2^{1/2} w \right\| - \zeta_2 \tag{2.4.21}$$

$$\xi_i \geq 0,\ i = 1,..,m \tag{2.4.22}$$

$$\nu_i \geq 0,\ i = 1,2 \tag{2.4.23}$$

$$\zeta_i \geq 0,\ i = 1,2 \tag{2.4.24}$$

As before, this problem can be solved with the two-step iterative SOCP procedure. Imposing the generative prior with soft constraints ensures that, as the amount of training data increases, the data overwhelms the prior and the algorithm converges to the maximum-margin separating hyperplane.

2.5 Experiments

The experiments were designed both to demonstrate the usefulness of the proposed approach for incorporation of generative prior into discriminative classification, and to address a broader question by showing that it is possible to use an existing domain theory to aid in a classification task for which it was not specifically designed. In order to construct the generative prior, the generative LDA classifier was trained on the data from the training classification task to estimate the Gaussian location parameters $\widehat{\mu}_i, i = 1,2$, as described in Section 2. The compression function $\chi_i(v)$ is subsequently computed (also as described in Section 2), and is used to set the hyperprior parameters via $\mu_i^j := \chi_i(l_{i,test}^j), i = 1,2$. In order to apply a domain theory effectively to the task for which it was not specifically designed, the algorithm must be able to estimate

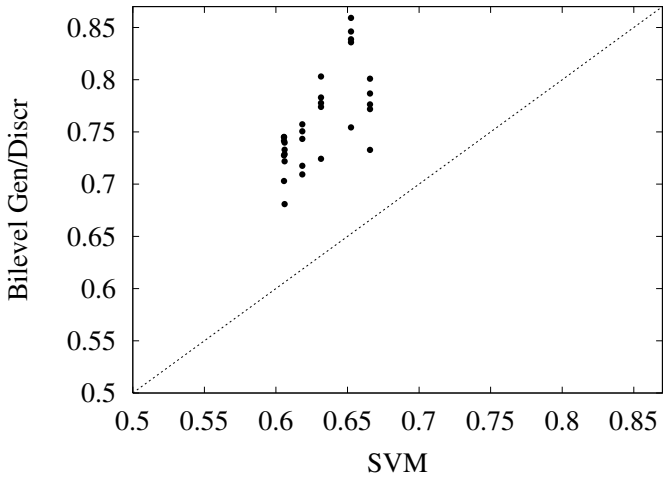


Figure 2.5.1: Performance of the bilevel discriminative classifier constrained by generative prior knowledge versus performance of SVM. Each point represents a unique pair of training/test tasks, with 0.5% of the test task data used for training. The results are averaged over 100 experiments.

its confidence in the decomposition of the domain theory with respect to this new learning task. In order to model the uncertainty in applicability of WordNet to newsgroup categorization, our system estimated its confidence in homogeneity of equivalence classes of semantic distances by computing the variance of each random variable $\chi_i(v)$ as follows: $\sigma_i(v) \triangleq \frac{\sum_{j:l^j_{i,train}=v} (\widehat{\mu^j_i} - \chi_i(v))^2}{|j:l^j_{i,tran}=v|}$. The hyperprior confidence matrices $\Omega_i, i = 1, 2$ were then reconstructed with respect to the test task semantic distances $l_{i,test}, i = 1, 2$ as follows: $[\Omega_i]_{j,k} := \begin{cases} \sigma_i(l^j_{i,test}), k = j \\ 0, k \neq j \end{cases}$. Identity matrices were used as covariance matrices of the lower-level prior: $\Sigma_1 = \Sigma_2 := I$. The rest of the parameters were set as follows: $\beta := 0.2, \varphi := 0.01$, $C_1 = C_2 := 1$, $C_3 := \infty$. These constants were chosen manually to optimize performance on Experiment 1 (for the training task: atheism vs. guns, test task: guns vs. mideast, see Figure 2.5.2) without observing any data from any other classification tasks.

The resulting classifier was evaluated in different experimental setups (with different pairs of newsgroups chosen for the training and the test tasks) to justify the following claims:

1. The bilevel generative/discriminative classifier with WordNet-derived prior knowledge has good low-sample performance, showing both the feasibility of automatically interpreting the knowledge embedded in WordNet and the efficacy of the proposed algorithm.

2. The bilevel classifier's performance improves with increasing training sample size.

3. Integrating generative prior into the discriminative classification framework results in better perfor-

mance than integrating the same prior directly into the generative framework via Bayes' rule.

4. The bilevel classifier outperforms a state-of-the-art discriminative multitask classifier proposed by Evgeniou and Pontil [34] by taking advantage of the WordNet domain theory.

In order to evaluate the low-sample performance of the proposed classifier, four newsgroups from the 20-newsgroup dataset were selected for experiments: *atheism, guns, middle east*, and *auto*. Using these categories, thirty experimental setups were created for all the possible ways of assigning newsgroups to training and test tasks (with a pair of newsgroups assigned to each task, under the constraint that the training and test pairs cannot be identical). In each experiment, we compared the following two classifiers:

1. Our bilevel generative-discriminative classifier with the knowledge transfer functions $\chi_i(v), \sigma_i(v), i = 1, 2$ learned from the labeled training data provided for the training task (using 90% of all the available data for that task). The resulting prior was subsequently introduced into the discriminative classification framework via our approximate bilevel programming approach

2. A vanilla SVM classifier which minimizes the regularized empirical risk:

$$\min_{w,b,\xi_i} \sum_{i=1}^{m} \xi_i + C_1 \|w\|^2 \tag{2.5.1}$$

$$\text{s.t.} y_i(w^T x_i + b) \geq 1 - \xi_i, i = 1, .., m \tag{2.5.2}$$

Both classifiers were trained on 0.5% of all the available data from the *test* classification task[3], and evaluated on the remaining 99.5% of the test task data. The results, averaged over one hundred randomly selected datasets, are presented in Figure 2.5.1, which shows the plot of the accuracy of the bilevel generative/discriminative classifier versus the accuracy of the SVM classifier, evaluated in each of the thirty experimental setups. All the points lie above the 45^o line, indicating improvement in performance due to incorporation of prior knowledge via the bilevel programming framework. The amount of improvement ranges from 10% to 30%, with all of the improvements being statistically significant at the 5% level.

The next experiment was conducted to evaluate the effect of increasing training data (from the test task) on the performance of the system. For this experiment, we selected three newsgroups (*atheism, guns,* and *middle east*) and generated six experimental setups based on all the possible ways of splitting these newsgroups into unique training/test pairs. In addition to the classifiers 1 and 2 above, the following classifiers were evaluated:

[3]SeDuMi software [68] was used to solve the iterative SOCP programs.

3. A state-of-the art multi-task classifier designed by Evgeniou and Pontil [34]. The classifier learns a set of related classification functions $f_t(x) = w_t^T x + b_t$ for classification tasks $t \in \{\text{training task, test task}\}$ given $m(t)$ data points $[x_{1t}, y_{1t}], .., [x_{m(t)t}, y_{m(t)t}]$ for each task t by minimizing the regularized empirical risk:

$$\min_{w_0, w_t, b_t, \xi_{it}} \sum_t \sum_{i=1}^{m(t)} \xi_{it} + \frac{C_1}{C_2} \sum_t \|w_t - w_0\|^2 + C_1 \|w_0\|^2 \tag{2.5.3}$$

$$\text{s.t. } y_{it}(w_t^T x_{it} + b_t) \geq 1 - \xi_{it}, i = 1, .., m(t), \forall t \tag{2.5.4}$$

$$\xi_{it} \geq 0, i = 1, .., m(t), \forall t \tag{2.5.5}$$

The regularization constraint captures a tradeoff between final models w_t being close to the average model w_0 and having a large margin on the training data. 90% of the training task data was made available to the classifier. Constant $C_1 := 1$ was chosen, and $C_2 := 1000$ was selected from the set $\{.1, .5, 1, 2, 10, 1000, 10^5, 10^{10}\}$ to optimize the classifier's performance on Experiment 1 (for the training task: atheism vs. guns, test task: guns vs. mideast, see Figure 2.5.2) after observing .05% of the test task data (in addition to the training task data).

4. The LDA classifier described in Section 2 trained on 90% of the *test* task data. Since this classifier is the same as the bottom-level generative classifier used in the bilevel algorithm, its performance gives an upper bound on the performance of the bottom-level classifier trained in a generative fashion.

Figure 2.5.2 shows performance of classifiers 1-3 as a function of the size of the training data from the test task (evaluation was done on the remaining test-task data). The results are averaged over one hundred randomly selected datasets. The performance of the bilevel classifier improves with increasing training data both because the discriminative portion of the classifier aims to minimize the training error and because the generative prior is imposed with soft constraints. As expected, the performance curves of the classifiers converge as the amount of available training data increases. Even though the constants used in the mathematical program were selected in a single experimental setup, the classifier's performance is reasonable for a wide range of data sets across different experimental setups, with the possible exception of Experiment 4 (training task: guns vs. mideast, testing task: atheism vs. mideast), where the means of the constructed elliptical priors are much closer to each other than in the other experiments. Thus, the prior is imposed with greater confidence than is warranted, adversely affecting the classifier's performance.

The multi-task classifier 3 outperforms the vanilla SVM by generalizing from data points across classification tasks. However, it does not take advantage of prior knowledge, while our classifier does. The

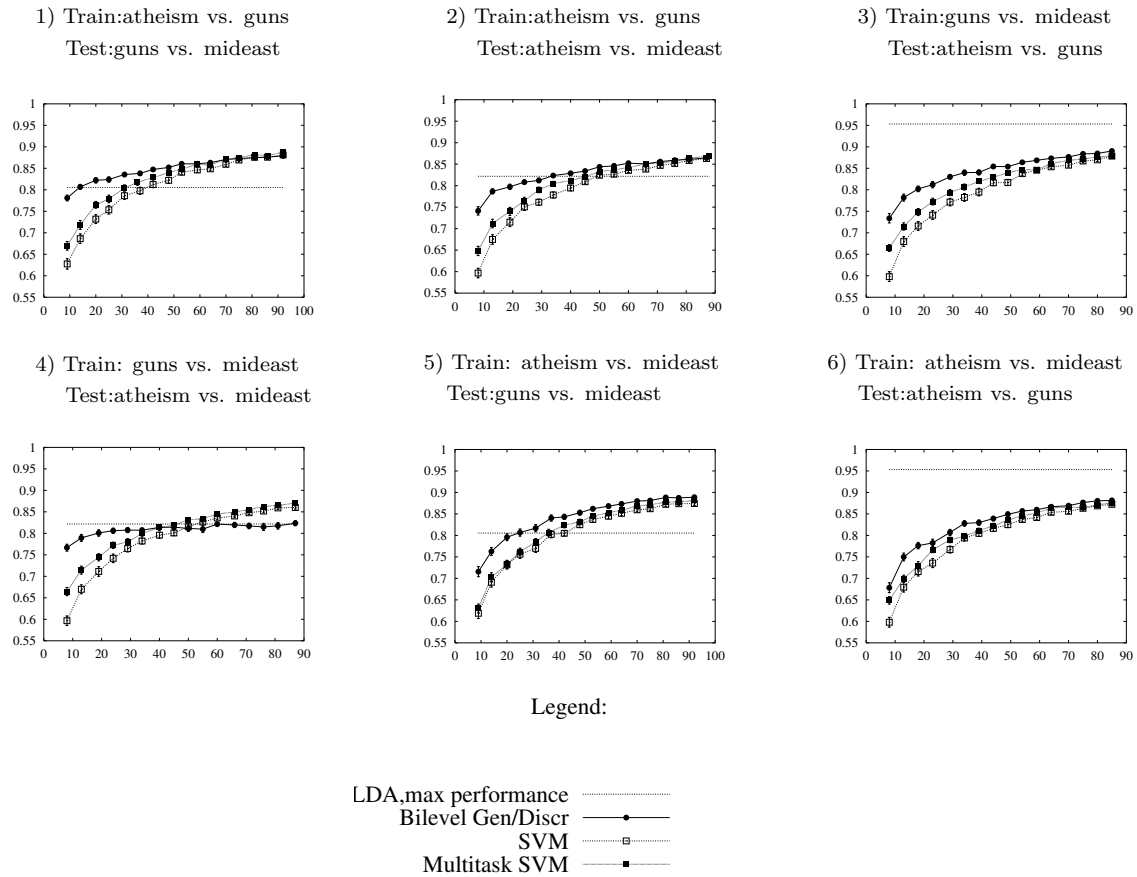
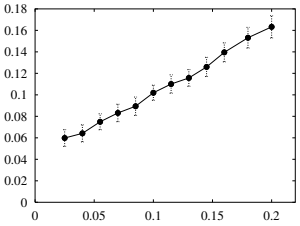


Figure 2.5.2: Test set accuracy as a percentage versus number of test task training points for two classifiers (SVM and Bilevel Gen/Discr) tested on six different classification tasks. For each classification experiment, the data set was split randomly into training and test sets in 100 different ways. The error bars based on 95% confidence intervals.

gain in performance of the bilevel generative/discriminative classifier is due to the fact that the relationship between the classification tasks is captured much better by WordNet than by simple linear averaging of weight vectors.

Because of the constants involved in both the bilevel classifier and the generative classifiers with Bayesian priors, it is hard to do a fair comparison between classifiers constrained by generative priors in these two frameworks. Instead, the generatively trained classifier 4 gives an empirical upper bound on the performance achievable by the bottom-level classifier trained generatively on the test task data. The accuracy of this classifier is shown as as a horizontal in the plots in Figure 2.5.2. Since discriminative classification is known to be superior to generative classification for this problem, the SVM classifier outperforms the generative classifier given enough data in four out of six experimental setups. What is more interesting, is that, for a range of training sample sizes, the bilevel classifier constrained by the generative prior outperforms both the

1) Accuracy as a function of β



2) Accuracy as a function of φ

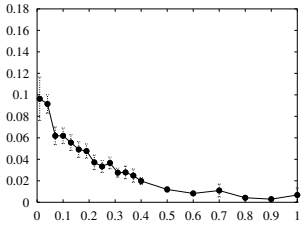


Figure 2.5.3: Plots of test set accuracy as percentage versus mathematical program parameter values. For each classification task, a random training set of size 9 was chosen from the full set of test task articles in 100 different ways. Error bars are based on 95% confidence intervals. All the experiments were performed on the training task: atheism vs. guns, test task: guns vs. mideast.

SVM trained on the same sample and the generative classifier trained on a much larger sample in these four setups. This means that, unless prior knowledge outweighs the effect of learning, it cannot enable the LDA classifier to compete with our bilevel classifier on those problems.

Finally, a set of experiments was performed to determine the effect of varying mathematical program parameters β and φ on the generalization error. Each parameter was varied over a set of values, with the rest of the parameters held fixed (β was increased up to its maximum feasible value). The evaluation was done in the setup of Experiment 1 (for the training task:atheism vs. guns, test task: guns vs. mideast), with the training set size of 9 points. The results are presented in Figure 2.5.3. Increasing the value of β is equivalent to requiring a hyperplane separator to have smaller error given the prior. Decreasing the value of φ is equivalent to increasing the confidence in the hyperprior. Both of these actions tighten the constraints (i.e., decrease the feasible region). With good prior knowledge, this should have the effect of improving generalization performance for small training samples since the prior is imposed with higher confidence. This is precisely what we observe in the plots of Figure 2.5.3.

2.6 Generalization Performance

Why does the algorithm generalize well for low sample sizes? In this section, we derive a theorem which demonstrates that the convergence rate of the generalization error of the constrained generative-discriminative classifier depends on the parameters of the mathematical program and not just the margin, as would be expected in the case of large-margin classification without the prior. In particular, we show that as the certainty of the generative prior knowledge increases, the upper bound on the generalization error of the classifier constrained by the prior decreases. By increasing certainty of the prior, we mean that either the hyper-prior becomes more peaked (i.e., the confidence in the locations of the prior means increases) or the

desired upper bounds on the Type I and Type II probabilities of error of the classifier decrease (i.e., the requirement that the lower-level discriminative player choose the restricted Bayes-optimal hyperplane is more strictly enforced).

The argument proceeds by bounding the fat-shattering dimension of the classifier constrained by prior knowledge. The fat-shattering dimension of a large margin classifier is given by the following definition [71]:

Definition 2.6.1. *A set of points $S = \{x^1 ... x^m\}$ is γ-shattered by a set of functions F mapping from a domain X to $\mathbb{R}$ if there are real numbers $r^1, ..., r^m$ such that, for each $b \in \{-1, 1\}^m$, there is a function f_b in F with $b(f_b(x^i) - r^i) \geq \gamma$, $i = 1..m$. We say that $r^1, ..., r^m$ witness the shattering. Then the fat-shattering dimension of F is a function $fat_F(\gamma)$ that maps γ to the cardinality of the largest γ-shattered set S.*

Specifically, we consider the class of functions

$$F = \{x \to w^T x : \|x\| \leq R, \|w\| = 1, \tag{2.6.1}$$

$$\frac{w^T(-\mu_1)}{\left\|\Sigma_1^{1/2} w\right\|} \geq \beta, \left\|\Omega_1^{-1/2}(\mu_1 - t_1)\right\| \leq \varphi, \frac{w^T \mu_2}{\left\|\Sigma_2^{1/2} w\right\|} \geq \beta, \left\|\Omega_2^{-1/2}(\mu_2 - t_2)\right\| \leq \varphi\}.$$

The following theorem bounds the fat-shattering dimension of our classifier:

Theorem 2.6.2. *Let F be the class of a-priori constrained functions defined by (2.6.1), and let $\lambda_{min}(P)$ and $\lambda_{max}(P)$ denote the minimum and maximum eigenvalues of matrix P, respectively. If a set of points S is γ-shattered by F, then $|S| \leq \frac{4R^2(\alpha^2(1-\alpha^2))}{\gamma^2}$, where $\alpha = \max(\alpha_1, \alpha_2)$ with $\alpha_1 = \min(\frac{\lambda_{min}(\Sigma_1)\beta}{\|\mu_2\|}, \frac{\|t_2\|^2 - (\lambda_{max}(\Omega_2)\varphi)^2}{\|t_2\|(\lambda_{max}(\Omega_2)\varphi)^2 + \|t_2\|)})$ and $\alpha_2 = \min(\frac{\lambda_{min}(\Sigma_1)\beta}{\|\mu_1\|}, \frac{\|t_1\|^2 - (\lambda_{max}(\Omega_1)\varphi)^2}{\|t_1\|((\lambda_{max}(\Omega_1)\varphi)^2 + \|t_1\|)})$, assuming that $\beta \geq 0$, $\|t_i\| \geq \|t_i - \mu_i\|$, and $\alpha_i \geq \frac{1}{\sqrt{2}}, i = 1, 2$.*

Proof. See Appendix A.2. □

We have the following corollary which follows directly from Taylor and Bartlett's [71] Theorem 1.5 and bounds the classifier's generalization error based on its fat-shattering dimension:

Corollary 2.6.3. *Let G be a class of real-valued functions. Then, with probability at least $1 - \delta$ over m independently generated examples z, if a classifier $h = sgn(g) \in sgn(G)$ has margin at least γ on all the examples in z, then the error of h is no more than $\frac{2}{m}(d * log(\frac{8em}{d}) log(32m) + log(\frac{8m}{\delta}))$ where $d_G = fat_G(\frac{\gamma}{16})$. If $G = F$ is the class of functions defined by (2.6.1), then $d_F \leq \frac{265R^2(4(\alpha^2(1-\alpha^2)))}{\gamma^2}$. If $G = F^*$ is the usual class of large margin classifiers (without the prior), then the result in [71] shows that $d_{F^*} \leq \frac{265R^2}{\gamma^2}$.*

Notice that both bounds depend on $\frac{R^2}{\gamma^2}$. However, the bound of the classifier constrained by the generative prior also depends on β and φ through the term $4(\alpha^2(1 - \alpha^2))$. In particular, as β increases, tightening

the constraints, the bound decreases, ensuring, as expected, quicker convergence of the generalization error. Similarly, decreasing φ also tightens the constraints and decreases the upper bound on the generalization error. For $\alpha > \frac{1}{\sqrt{2}}$, the factor $4(\alpha^2(1-\alpha^2))$ is less than 1 and the upper bound on the fat-shattering dimension d_F is tighter than the usual bound in the no-prior case on d_{F^*}.

Since β controls the amount of deviation of the decision boundary from the Bayes-optimal hyperplane and φ depends on the variance of the hyper-prior distribution, tightening of these constraints corresponds to increasing our confidence in the prior. Note that a high value β represents high level of user confidence in the generative elliptical model. Also note that there are two ways of increasing the tightness of the hyperprior constraint (2.4.7) - one is through the user-defined parameter φ, the other is through the automatically estimated covariance matrices $\Omega_i, i = 1, 2$. These matrices estimate the extent to which the equivalence classes defined by WordNet create an appropriate decomposition of the domain theory for the newsgroup categorization task. Thus, tight constraint (2.4.7) represents both high level of user confidence in the means of the generative classification model (estimated from WordNet) and a good correspondence between the partition of the words imposed by the semantic distance of WordNet and the elliptical generative model of the data. As φ approaches zero and β approaches its highest feasible value, the solution of the bilevel mathematical program reduces to the restricted Bayes optimal decision boundary computed solely from the generative prior distributions, without using the data.

Hence, we have shown that, as the prior is imposed with increasing level of confidence (which means that the elliptical generative model is deemed good, or the estimates of its means are good, which in turn implies that the domain theory is well-suited for the classification task at hand), the convergence rate of the generalization error of the classifier increases. Intuitively, this is precisely the desired effect of increased confidence in the prior since the benefit derived from the training data is outweighed by the benefit derived from prior knowledge. For low data samples, this should result in improved accuracy assuming the domain theory is good, which is what the plots in Figure 2.5.3 show.

2.7 Comparison between the Vapnik-Chervonenkis (VC) framework and the Fat-shattering framework

The contents of this section are technical and may be skipped without loss of continuity. We show that the fat-shattering framework provides justification for stronger prior knowledge than the traditional VC framework. In particular, we show that the VC dimension of our bilevel classifier can reman large even when its hypothesis space is severely constrained by prior knowledge. The fat shattering dimension, however,

continues to decrease, justifying the use of stronger prior knowledge. This argument appears in our paper [29] which explores the effects of rotational constraints on the normal of the separating hyperplane.

To make the argument precise, let us define the VC dimension of a classifier which governs its generalization performance in the classical Vapnik-Chervonenkis framework [76]:

Definition 2.7.1. A set of points $S = \{x^1...x^m\}$ is shattered by a set of functions F mapping from a domain X to $\{-1, 1\}$ if, for each $b \in \{-1, 1\}^m$, there is a function f_b in F with $bf_b(x^i) = 1$, $i = 1..m$. The VC-dimension of F is the cardinality of the largest shattered set S.

In this section, we consider a simpler class of functions than in Section 2.6, defined as:

$$F' = \{x \rightarrow w^T x : \|x\| \leq R, \|w\| = 1, w^T v \geq \rho, \|v\| = 1\} \tag{2.7.1}$$

Class F' is a special case of the class F defined by 2.6.1, with $\Sigma_1 = \Sigma_2 = I$, $v = t_2 - t_1$, $\rho = 2\beta$, and $\phi = 0$ (Ω_1 and Ω_2 arbitrary).

It is well-known that the VC-dimension of F' in $\mathbb{R}^n$ is $n+1$ when $\rho = -1$ (i.e., unconstrained hyperplane classifier - see, e.g., [7]). Interestingly, the VC-dimension of *constrained* F' is at least n with any constraint imposed on $w \in W$ as long as there is an open subset of W that satisfies the constraints (this result follows from [33]). This means that any value of ρ in the conic constraint cannot result in significant improvement in the classifier's generalization ability as measured by its VC-dimension. The following theorem shows that the VC-dimension of a relatively weakly constrained classifier achieves this lower bound of n:

Theorem 2.7.2. *For the class* $F_C = \{x \rightarrow sign(\sum_{i=1}^{n} \omega_i x_i + \theta) : \omega_1 > 0\}$, *VC-dimension of* $F_C = n$.

Proof. The proof uses techniques from [7]. Let $F_C = \{x \rightarrow sign(\omega_1 x_1 + \overline{\omega}^T \overline{x} + \theta) : \omega_1 > 0\}$, where $\overline{x} = [x_2, ..., x_n]^T$ is the projection of x into the hyperplane $\{\omega_1 = 0\}$ and $\overline{\omega} = [\omega_2, ...\omega_n]^T$.

First, observe that $\{\omega_1 > 0\}$ defines an open subset of W. Hence, the VC-dimension of F_C is at least n. Now, we show by contradiction that a set of $n + 1$ points cannot be shattered by F_C. Assume that some set of points $x^1, ..., x^{n+1} \in \mathbb{R}^n$ can be shattered. Let $\overline{x^1}, ..., \overline{x^{n+1}} \in \mathbb{R}^{n-1}$ be their projections into the hyperplane $\{\omega_1 = 0\}$. There are two cases:

Case 1: $\overline{x^1}, ..., \overline{x^{n+1}}$ are distinct. Since these are $n + 1$ points in an $(n - 1)$-dimensional hyperplane, by Radon's Theorem [41] they can be divided into two sets S_1 and S_2 whose convex hulls intersect. Thus, $\exists \lambda_i, \lambda_j (0 \leq \lambda_i, \lambda_j \leq 1)$

$$\text{such that} \quad \sum_{i\,:\,\overline{x^i} \in S_1} \lambda_i \overline{x^i} = \sum_{j\,:\,\overline{x^j} \in S_2} \lambda_j \overline{x^j} \tag{2.7.2}$$

$$\text{and} \quad \sum_{i\,:\,\overline{x^i} \in S_1} \lambda_i = \sum_{j\,:\,\overline{x^j} \in S_2} \lambda_j = 1 \tag{2.7.3}$$

Since $x^1, ..., x^{n+1}$ are shattered in $\mathbb{R}^n$, $\exists \omega_1, \overline{\omega}, \theta$ such that $\omega_1 x_1^i + \overline{\omega}^T \overline{x^i} \geq \theta$ for all $\overline{x^i} \in S_1$. Multiplying by λ_i and summing over i, we get (after applying (7))

$$\overline{\omega}^T \sum_{i\,:\,\overline{x^i} \in S_1} \lambda_i \overline{x^i} \geq \theta - \omega_1 \sum_{i\,:\,\overline{x^i} \in S_1} \lambda_i x_1^i \tag{2.7.4}$$

Similarly, for all $\overline{x^j} \in S_2$, $\omega_1 x_1^j + \overline{\omega}^T \overline{x^j} < \theta \Rightarrow$

$$\overline{\omega}^T \sum_{j\,:\,\overline{x^j} \in S_2} \lambda_j \overline{x^j} < \theta - \omega_1 \sum_{j\,:\,\overline{x^j} \in S_2} \lambda_j x_1^j \tag{2.7.5}$$

$$\text{Combining (8), (9), and (6) yields } \omega_1 \Big(\sum_{j\,:\,\overline{x^j} \in S_2} \lambda_j x_1^j - \sum_{i\,:\,\overline{x^i} \in S_1} \lambda_i x_1^i \Big) < 0 \tag{2.7.6}$$

$$\text{Since } \omega_1 > 0, \Big(\sum_{j\,:\,\overline{x^j} \in S_2} \lambda_j x_1^j - \sum_{i\,:\,\overline{x^i} \in S_1} \lambda_i x_1^i \Big) < 0 \tag{2.7.7}$$

Now, shattering the same set of points, but reversing the labels of S_1 and S_2 implies that $\exists \omega_1^{'}, \overline{\omega^{'}}, \theta^{'}$ such that $\omega_1^{'} x_1^i + \overline{\omega^{'}}^T \overline{x^i} < \theta^{'}$ for all $\overline{x^i} \in S_1$ and $\omega_1^{'} x_1^j + \overline{\omega^{'}}^T \overline{x^j} \geq \theta^{'}$ for all $\overline{x^j} \in S_2$. An argument identical to the

one above shows that

$$\omega_1^{'}(\sum_{j\,:\,\overline{x^j} \in S_2} \lambda_j x_1^j - \sum_{i\,:\,\overline{x^i} \in S_1} \lambda_i x_1^i) > 0 \tag{2.7.8}$$

$$\text{Since } \omega_1^{'} > 0, \text{ (} \sum_{j\,:\,\overline{x^j} \in S_2} \lambda_j x_1^j - \sum_{i\,:\,\overline{x^i} \in S_1} \lambda_i x_1^i) > 0, \text{ which contradicts (11)}$$

Case 2: Two distinct points x^1 and x^2 project to the same point $\overline{x^1} = \overline{x^2}$ (13) on the hyperplane $\{\omega_1 = 0\}$. Assume, wlog, that $x_1^1 < x_1^2$ (14). Since x^1 and x^2 are shattered, $\exists \omega_1, \overline{\omega}, \theta$ such that $\omega_1 x_1^1 + \overline{\omega}^T \overline{x^1} \geq \theta > \omega_1 x_1^2 + \overline{\omega}^T \overline{x^2}$, which, together with (13) and (14), implies that $\omega_1 < 0$, a contradiction. □

This result means that using $\rho = 0$ in the conic constraint is sufficient to achieve the maximum theoretical improvement within the VC framework[4]. However, it is unsatisfactory in a sense that it contradicts our intuition (and empirical results) which suggests that stronger prior knowledge should help the classifier reduce its generalization error faster. The following theorem shows that the fat-shattering dimension decreases continuously with increasing ρ in the conic constraint, giving us the desired guarantee. Technically, the fat-shattering dimension is a function of the margin γ, so we use the following definition of function domination to specify what we mean by decreasing fat-shattering dimension:

Definition 2.7.3. A function $f_1(x)$ is dominated by a function $f_2(x)$ if, for all x, $f_1(x) \leq f_2(x)$ and, at least for one a, $f_1(a) < f_2(a)$. When we say that $f_\rho(x)$ decreases with increasing ρ, we mean that $\rho_1 < \rho_2$ implies that $f_{\rho_2}(x)$ is dominated by $f_{\rho_1}(x)$.

Theorem 2.7.4. *For the class* $F_{v,\rho} = \{x \to \omega^T x + \theta : \|\omega\|_2 = 1, \|v\|_2 = 1, \|x\|_2 \leq R, \omega^T v > \rho \geq 0\}$, $fat_{F_{v,\rho}}(\gamma)$ *decreases with increasing* ρ.

Proof. The fat-shattering dimension obviously cannot increase with increasing ρ, so we only need to find a value of γ where it decreases. We show that this happens at $\gamma' = R\sqrt{1-\rho_2^2}$. First, we upper bound $\text{fat}_{F_{v,\rho_2}}(\gamma')$ by showing that, in order to γ'-shatter two points, the separating hyperplane must be able to rotate through a larger angle than that allowed by the constraint ${\omega^1}^T v > \rho_2$. Assume that two points x^1, x^2 can be γ'-shattered by F_{v,ρ_2}. Then $\exists \omega^1, \omega^2, \theta^1, \theta^2, r^1, r^2$ such that ${\omega^1}^T x^1 + \theta^1 - r^1 \geq \gamma'$, ${\omega^1}^T x^2 + \theta^1 - r^2 \leq -\gamma'$, ${\omega^2}^T x^1 + \theta^2 - r^1 \leq -\gamma'$, ${\omega^2}^T x^2 + \theta^2 - r^2 \geq \gamma'$. Combining the terms and applying the Cauchy-Schwartz inequality, we get $\left\|\omega^1 - \omega^2\right\| \geq \frac{2\gamma'}{R}$. Squaring both sides, expanding $\left\|\omega^1 - \omega^2\right\|^2$ as $\left\|\omega^1\right\|^2 + \left\|\omega^2\right\|^2 - 2{\omega^1}^T \omega^2$,

[4]The constraint $\{w_1 > 0\}$ is weak since it only cuts the volume of the hypothesis space by $\frac{1}{2}$.

and using the fact that $\left\|\omega^1\right\| = \left\|\omega^2\right\| = 1$ yields

$$\omega^{1^T}\omega^2 \leq 1 - \frac{2\gamma'^2}{R^2} = 2\rho_2^2 - 1 \tag{15}$$

Since the angle between ω^1 and ω^2 cannot exceed the sum of the angle between ω^1 and the prior v and the angle between v and ω^2, both of which are bounded above by $\arccos(\rho_2)$, we get (after some algebra) $\omega^{1^T}\omega^2 > 2\rho_2^2 - 1$, which contradicts (15).

$$\text{Thus, } fat_{F_{v,\rho_2}}(R\sqrt{1-\rho_2^2}) < 2 \tag{16}$$

Now, we lower bound $\text{fat}_{F_{v,\rho_1}}(\gamma')$ by exhibiting two points γ'-shattered by F_{v,ρ_1}. Wlog, let $v = [0, 1, 0, ..0]^T$. It is easy to verify that $x^1 = [R, 0, ..0]^T$ and $x^2 = [-R, 0, ..0]^T$ can be $R\sqrt{1-\rho_2^2}$-shattered by F_{v,ρ_1}, witnessed by $r^1 = r^2 = 0$.

$$\text{Hence, } fat_{F_{v,\rho_1}}(R\sqrt{1-\rho_2^2}) \geq 2 \tag{17}$$

which, combined with (16), completes the argument. □

The result of Theorem 2.7.2 is important because it shows that even weak prior knowledge improves the classifier's generalization performance in the VC framework which makes less assumptions about the data than the fat-shattering framework. However, it is the result of Theorem 2.7.4 within the fat-shattering framework which justifies the use of stronger prior knowledge.

2.8 Related Work

There are a number of approaches to combining generative and discriminative models. Several of these focus on deriving discriminative classifiers from generative distributions [74, 73] or on learning the parameters of generative classifiers via discriminative training methods [40, 63]. The closest in spirit to our approach is the Maximum Entropy Discrimination framework [46, 45], which performs discriminative estimation of parameters of a generative model, taking into account the constraints of fitting the data and respecting the prior. One important difference with our framework is that, in estimating these parameters, maximum entropy discrimination minimizes the distance between the generative model and the prior, subject to satisfying the discriminative constraint that the training data be classified correctly with a given margin. Our framework, on the other hand, maximizes the margin on the training data subject to the constraint that the generative model is not too far from the prior. This emphasis on maximizing the margin allows us to derive a-priori

bounds on the generalization error of our classifier based on the confidence in the prior which are not (yet) available for the maximum entropy framework. Another difference is that our approach performs classification via a single generative model, while maximum entropy discrimination averages over a set of generative models weighted by their probabilities. This is similar to the distinction between maximum-a-posteriori and Bayesian estimation and has repercussions for tractability. Maximum entropy discrimination, however, is more general than our framework in a sense of allowing a richer set of behaviors based on different priors.

Ng et al. [61, 58] explore the relative advantages of discriminative and generative classification and propose a hybrid approach which improves classification accuracy for both low-sample and high-sample scenarios. Collins [22] proposes to use the Viterbi algorithm for HMMs for inferencing (which is based on generative assumptions), combined with a discriminative learning algorithm for HMM parameter estimation. These research directions are orthogonal to our work since they do not explicitly consider the question of integration of prior knowledge into the learning problem.

In the context of support vector classification, various forms of prior knowledge have been explored. Scholkopf et al. [65] demonstrate how to integrate prior knowledge about invariance under transformations and importance of local structure into the kernel function. Fung et al. [38] use domain knowledge in form of labeled polyhedral sets to augment the training data. Wu and Srihari [79] allow domain experts to specify their confidence in the example's label, varying the effect of each example on the separating hyperplane proportionately to its confidence. Sun and DeJong [69] propose an algorithm which uses domain knowledge (such as WordNet) to identify relevant features of examples and incorporate resulting information in form of soft constraints on the hypothesis space of SVM classifier. Mangasarian et al. [54] suggest the use of prior knowledge for support vector regression. In all of these approaches, prior knowledge takes the form of explicit constraints on the hypothesis space of the large-margin classifier. In this work, the emphasis is on generating such constraints automatically from domain knowledge interpreted in the generative setting. As we demonstrate with our WordNet application, generative interpretation of background knowledge is very intuitive for natural language processing problems.

Second-order cone constraints have been applied extensively to model probability constraints in robust convex optimization [51, 9] and constraints on the distribution of the data in minimax machines [49, 44]. Our work, as far as we know, is the first one which models prior knowledge with such constraints. The resulting optimization problem and its connection with Bayes optimal classification is very different from the approaches mentioned above.

Our work is also related to empirical Bayes estimation [18]. In empirical Bayes estimation, the hyper-prior parameters of the generative model are estimated using statistical estimation methods (usually maximum

likelihood or method of moments) through the marginal distribution of the data, while our approach learns those parameters discriminatively using the training data.

2.9 Conclusions

In this chapter, we studied a problem in which the expert's domain theory (in form of an ontology) was not specifically designed for the text classification task at hand. We presented a method for constructing a Bayesian prior from this more intuitive specification of the domain knowledge. We also presented an efficient learning algorithm for integrating this prior with new evidence from the training data. Our algorithm explores two issues which the Bayesian framework glosses over. The first is utilizing existing domain knowledge in form intuitive for domain experts to specify. Second is efficient inferencing based on this form of prior knowledge and new empirical evidence.

Chapter 3

Prior Knowledge and Reinforcement Learning

In this chapter, we explore the use of prior knowledge in reinforcement learning systems. When a reinforcement learning agent is provided with a description of its environment in form of precise transition probabilities and rewards, the optimal policy can be determined offline without any interaction with the environment. Algorithms such as policy and value iteration [70] can be used to compute the optimal solution, which may be subsequently applied online. Unfortunately, in many domains it is unrealistic to expect that an expert will be able to come up with precise system dynamics. In such domains, an agent can resort to reinforcement learning (RL) to explore its environment. However, extensive exploration can be undesirable for the following reasons: it is time-consuming, expensive (in terms of wear and tear on the robotic equipment), and perilous when the agent chooses to explore dangerous states (e.g., nuclear reactor meltdown for an agent controlling a nuclear plant or car going off the road for a car-driving agent. Abbeel and Ng also describe a helicopter crash which occurred during overly aggressive exploration[4, 3]).

In this chapter, we describe two strategies for combining prior knowledge which defines important properties of an MDP with limited online exploration. In both cases, the system which has the benefit of prior information is able to discover the optimal policy with significantly less exploration than the standard RL system which starts out with a blank slate. The strategy described in Section 3.1 (which we call qualitative reinforcement learning) enables domain experts to define MDPs in terms of qualitative statements such as "a higher mountain is more difficult to climb than a lower mountain". We define the semantics of such qualitative statements in terms of constraints on transition probabilities and introduce a qualitative planning algorithm for solving qualitative MDPs. When qualitative MDP descriptions lead to ambiguous solutions, a variation of our algorithm uses limited exploration to correct the qualitative descriptions and disambiguate the qualitative planning results.

In Section 3.2, we describe an alternative approach (which we call active reinforcement learning) which requires an expert to provide the agent with a precisely defined but possibly inaccurate MDP. While the optimal policy can be determined for this model via policy iteration, the agent does not immediately apply it online. Instead, the agent determines the sensitivity of this optimal policy to changes in transitions and

rewards of the MDP. It then focuses its exploration on the regions of space to which the optimal policy is most sensitive.

We present experimental evidence that both approaches reduce the amount of exploration necessary to determine optimal policies. We also present theoretical convergence guarantees for both algorithms.

3.1 Qualitative Reinforcement Learning

One reason why excessive exploration is wasteful is due to the agent's inability to infer how the world works in new states based on its experiences in previously visited states. Consider, for example, a car driving agent which drives off the road because it is going too fast while taking a turn. Even if the agent learns that the optimal policy in this situation is to slow down, it will repeat the same mistake when taking a similar turn at an even faster speed. This inability of the agent to transfer acquired information between states does not just increase the amount of exploration required to learn a good policy - it also prevents the agent from acting optimally in parts of the environment unseen during the learning stage (a car driving agent that learns to drive on small hills may have trouble after being transferred to a mountainous terrain, even though the same principles apply).

Notice that, in the above example, simple qualitative statements about the domain of the sort: "a higher mountain is more difficult to climb than a lower mountain", or "a turn is easier to take at a lower speed" may be sufficient to facilitate the kind of reasoning needed to generalize the learning experience and enable the agent to solve the problem without resorting to extensive exploration. However, these statements cannot be expressed in the language of MDP transition probabilities, and can never be fully acquired through reinforcement learning unless one sees every mountain in the world (some of which may be too dangerous to climb).

In this chapter, we introduce a framework which allows the expert to specify a set of comparative statements about the domain. This qualitative description of the problem is satisfied by multiple quantitative worlds, with each world describing an MDP with completely specified transition probabilities and rewards. We present an algorithm which, given such a qualitative description, returns a set of policies guaranteed to contain the optimal policy for every possible quantitative instantiation of the description. As an example, we apply our algorithm to the well-known problem of driving a car up a steep mountain [70], *with the caveat that the output of the car's engine is corrupted by arbitrary bounded stochastic noise.* Since the optimal policy depends on the engine power, our algorithm can be viewed as a tool for examining the sensitivity of the optimal policy to noise. We also apply the algorithm to another well-studied problem, that of balancing

a pole on a cart [70], under a similar assumption that the power of the cart's engine is uncertain.

Our algorithm allows MDP designers to obtain optimal solutions to problems without having to provide completely quantified specifications if the set of policies it returns is small enough to achieve good behavior in most states. If that is not the case, we present a variant of the algorithm which, given a qualitative description of the problem, combines it with limited exploration to discover the optimal policy much faster than traditional reinforcement learning, and the policy that it discovers is more broadly applicable.

The rest of the section is organized as follows: we describe related work in Subsection 3.1.1. In Subsection 3.1.2, we describe the variant of MDPs which we study. In Subsections 3.1.3 and 3.1.4 , our framework of qualitative MDPs (QMDP) and qualitative reinforcement learning (QRL) is described. Experiments are presented in Subsection 3.1.5, followed by conclusions in Subsection 3.1.6. The work described in this section has been published [31], but without the full proofs. Convergence guarantees in Appendix B.3 are previously unpublished.

3.1.1 Related Work

Qualitative Markov Decision Processes have been studied by Bonet and Pearl [13] and Sabbadin [64]. Bonet's study is purely theoretical, Sabbadin describes an application of his algorithm to a 3×3 gridworld. By contrast, we describe experiments with our algorithm on much more realistic problems which are an order of magnitude bigger than the 3×3 gridworld. More importantly, there is no clear connection between qualitative representations of MDPs proposed in these two papers and quantitative probabilities which can be estimated via empirical interaction with the environment. For this reason, neither study attempts to combine the qualitative problem description with quantitative exploration. In our approach, qualitative statements have a clear probabilistic interpretation, which enables us to construct such a combination.

Several ways of limiting exploration in reinforcement learning with prior knowledge have been proposed. Shaping [57, 50] attempts to direct the agent to explore regions which are likely to lead to good solutions by modifying the reward function. However, it may be difficult to determine a-priori which states will ultimately lead to good solutions and which should not be explored. Apprenticeship learning [4, 3] is a framework in which an agent learns the expert's reward function by observing his demonstrated behavior, thus avoiding direct interaction with the environment. The advantage of our approach is that our model of prior knowledge only requires specifying how the world works, not how to explore it. It may be used in domains where the expert has pertinent information about the world dynamics, but does not know how to solve the problem.

An alternative approach to dealing with uncertainty in the specification of MDPs without resorting to exploration is the minimax robustness framework (see e.g.,[39]). In this framework, the agent is also

presented with a description of the world which corresponds to a set of completely specified MDPs. The agent's goal is to select the best optimal policy, knowing that for any policy the agent selects, adversarial nature will choose the worst possible world in which to evaluate it. In our framework, on the other hand, the agent seeks a set of policies which contains the optimal one for every possible completely specified MDP.

3.1.2 Preliminaries

Instead of regular Markov Decision Processes, in what follows we use a variant of MDPs in which the agent is only interested in the nearest reward. A reward received later is forsaken for any probability of receiving any reward earlier, and bigger expected rewards are preferred to smaller rewards received at the same time. Ties between rewards to be received after n steps are broken by looking at rewards to be received after $n+1$ steps, once again preferring bigger rewards to smaller, and so on, up to N steps ahead. We will refer to this MDP as Myopic MDP because of its strong preference for receiving rewards sooner. In Section 3.1.5, we present experimental evidence that this variant gives reasonable policies for control problems.

In order to formalize this paradigm, we use the framework of generalized Markov Decision Processes. A generalized finite Markov Decision Process is a tuple $(S, A, P, R, \otimes, \oplus, Next)$, where S is a finite set of states, A is a finite set of actions, R is a reward function, $P : S'|S\text{x}A \to [0, 1]$ is a transition probability function, $Next : S\text{x}A \to S$ is a set of states reachable with nonzero probability in one step after taking action $a \in A$ in state $s \in S$, a summary operator $\oplus$ defines the value of transitions based on the value of the successor states, and a summary operator $\otimes$ defines the value of a state based on the values of all state-action pairs. These operators are used to define the generalized form of Bellman's equation as follows: for each state s, the optimal value function $V^*(s) = [H[KV^*]](s)$, where $[KV](s, a) = R(s, a) + \oplus_{s'}^{(s,a)} V(s')$ and $[HV](s) = \otimes_a^{(s)}([KV](s, a))$. Setting $\oplus_{s'}^{(s,a)} g(s') = \alpha \sum_{s'} P(s'|s, a)g(s')$ and $\otimes_a^{(s)} f(s, a) = \max_a f(s, a)$ recovers the conventional MDP formulation with the discount factor $\alpha \in (0, 1)$. In our myopic framework, on the other hand, the value function $V_\pi(s)$ for a fixed policy $\pi : S \to A$ is a N-dimensional vector, with each component $V_{\pi,i}, i \in 1, .., N$ representing the expected positive reward the agent will receive i steps after starting out in state s and following π[1]. Similarly, the reward $R(s, a) = [r(s, a), 0, 0, .., 0]$ is a reward vector indicating that reward $r(s, a) \geq 0$ is received for choosing action a in state s. The summary operators are defined to facilitate correct propagation of rewards: $\oplus_{s'}^{(s,a)} g(s') = \sum_{s'} P(s'|s, a)[0, g(s')]$ propagates rewards back from the successor states. Before we define the $\otimes$ operator, we need to impose an order relation on the values of states. This is done with the following definition:

[1] N is the horizon of the MDP. All of our results assume that N is large enough and still hold as $N \to \infty$.

Definition 3.1.1. Let $U \in \mathbb{R}^n$ and $V \in \mathbb{R}^n$ be two componentwise non-negative n-dimensional vectors.

$$\text{Let } f^*(U,V) = \min_{i:U_i>V_i\geq 0 \text{ or } V_i>U_i\geq 0} i$$

be the smallest component, the value of which is strictly greater in one of the vectors than in the other one. Then define $U \prec V \Leftrightarrow \{(f^* \text{ exists }) \wedge (U_{f*} < V_{f^*}))$, $U = V \Leftrightarrow f^*$ does not exist, and $U \preceq V \Leftrightarrow \{U \prec V \vee U = V\}$.

This order instantiates the myopic comparison of values of two actions. If $U(s)$ and $V(s)$ represent the values of two policies executed in s, then $f^*(U(s), V(s))$ is the first time step in which expected rewards of following U and V differ, and $\prec$ prefers the policy with larger reward in this time step. It can be verified that $\preceq$ is a total order (see Thm B.1.1 in the Appendix), which means that a maximum is well-defined for any finite set of vectors. We take the $\otimes$ operator to be this maximum: $\otimes_a^{(s)} f(s,a) = max_a^{\preceq} f(s,a)$.

The optimal value function $V^*(s)$ which satisfies Bellman's equation can be computed via value iteration, which successively updates

$$V^{t+1}(s) = [H[KV^t]](s) \tag{3.1.1}$$

until convergence or via policy iteration, which consists of policy evaluation (which computes

$$V_\pi^{t+1}(s) = [KV_\pi^t](s, \pi(s)) \tag{3.1.2}$$

for a fixed policy π) interleaved with policy improvement, which updates

$$V^{t+1}(s) = [HV_\pi^t](s) \tag{3.1.3}$$

The following theorem shows that for the Myopic MDP, policy evaluation and policy iteration converge:

Theorem 3.1.2. *For the Myopic MDP, there is a unique optimal value function V^* which satisfies the myopic Bellman's equation. Policy iteration converges to V^*. Moreover, for any fixed policy π, there is a unique optimal value function V_π^* which satisfies $V_\pi^* = KV_\pi^*$, and policy evaluation converges to V_π^*.*

Proof. (sketch, see Appendix, Theorem B.1.3 for the complete proof.) The proof is based on showing equivalence between the Myopic MDP policy iteration and policy iteration for a conventional MDP with sufficiently low discount factor α. □

In order to model qualitative knowledge, we rely on the notion of first-order stochastic dominance [67], defined as:

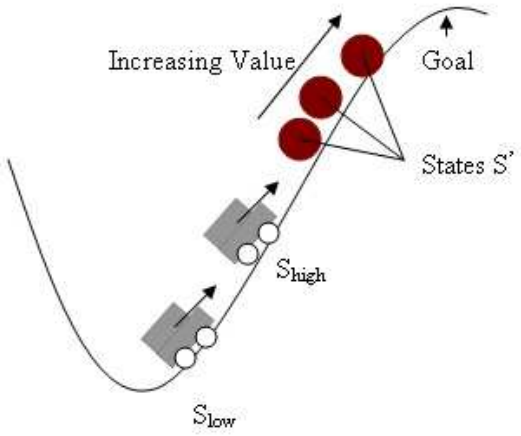


Figure 3.1.1: Mountain Car Domain

Definition 3.1.3. Let $G = \{g_1, ..., g_n\}$ be a support set for probability distributions P_1 and P_2. Let O be a partial order relation on G and define $\overline{O}(y) = \{x \in G : yOx\}$ to be the set of elements of G at least as good as y according to O (similarly, $\underline{O}(y) = \{x \in G : xOy\}$ is the set of elements no better than y according to O).

We say that P_1 stochastically dominates P_2 with respect to O if $\forall y \in G$, $P_1(\overline{O}(y)) \geq P_2(\overline{O}(y))$ where $P(S) = \sum_{s \in S} P(s)$ is the probability of a set. If, in addition, $\exists z \in G : P_1(\overline{O}(z)) > P_2(\overline{O}(z))$, we say that P_1 strictly stochastically dominates P_2 with respect to O (e.g., using $\leq$ for O gives first-order stochastic dominance on the real line).

3.1.3 Qualitative MDP

The policy evaluation step of the policy iteration algorithm performed via dynamic programming (Equation (3.1.2)) has the following useful monotonicity property when applied to Myopic MDPs:

Lemma 3.1.4. *Suppose $V^0 = \mathbf{0}$. Let s_1 and s_2 be any two states. If $V^t(s_1) \prec V^t(s_2)$, then for all subsequent iterations $q > t$ of policy evaluation, $V^q(s_1) \prec V^q(s_2)$.*

Proof. By induction on t, for any s and for any $j \leq t$, $V^t_j(s)$ (the component of $V^t(s)$ which represents the discounted expected reward received after j steps) does not change after step t (see Proposition B.1.2 in the Appendix for a proof of this fact). Moreover, $V^t_j(s) = 0$ for any $j > t$. If $V^t(s_1) \prec V^t(s_2)$, then by Definition 3.1 $\exists f^* : (V^t_f(s_1) = V^t_f(s_2), \forall f < f^*) \wedge (V^t_{f^*}(s_1) < V^t_{f^*}(s_2))$. Since $V^t_j(s_1) = 0$ for any $j > t$, $f^* \leq t$. $\square$

SameOrder(Oracle, $Order, s_1, s_2, \Pi_1, \Pi_2$)
Input: Procedure Oracle, $Order$ on S, States $s_1 \in S, s_2 \in S$, Sets of actions Π_1, Π_2
Output: $order \in \{'<', '>', '=', '?'\}$
1. **if** Oracle($Order, s_1, a_1, s_2, a_2$) returns the same value $order$ for all pairs of actions $a_1, a_2 \in \Pi_1 \times \Pi_2$
2. **then return** $order$
3. **else return**'?'

Policy Evaluation(Set of policies Π)
1. $j \leftarrow 0$
2. **for all** pairs of states $s_1, s_2 \in S \times S$
3. **do** $Step\_Order^j \leftarrow$ SameOrder (Reward_Oracle,$\varnothing$, s_1, s_2,$\Pi(s_1)$,$\Pi(s_2)$)
4. **repeat**
5. **for all** pairs of states $s_1, s_2 \in S \times S$
6. **do** $order \leftarrow$SameOrder (Oracle, $Step\_Order^j$,s_1,s_2,$\Pi(s_1)$,$\Pi(s_2)$)
7. $Step\_Order^{j+1}(s_1, s_2) \leftarrow order$
8. **if** $Order(s_1, s_2) = '='$
9. **then** $Order(s_1, s_2) \leftarrow order$
10. $j \leftarrow j + 1$
11. **until** $Order$ stops changing
12. **return** $Order$

Policy Improvement($Order$ on S)
1. **for all** states $s \in S$
2. **do** $best\_actions \leftarrow \varnothing$
3. **for all** actions $a, a' \in A x best\_actions$
4. **do if** Oracle($Order, s, a, s, a'$) $= '>'$
5. **then** $best\_actions \leftarrow \{best\_actions \cup \{a\}\} \backslash \{a'\}$
6. **if** Oracle($Order, s, a, s, a'$) $= '?'$
7. **then** $best\_actions \leftarrow best\_actions \cup \{a\}$
8. $\Pi(s) \leftarrow \varnothing$
9. **for all** actions $a \in best\_actions$
10. **do** $\Pi(s) \leftarrow \Pi(s) \cup \{a\}$
11. **return** Π

Policy Iteration()
1. Select arbitrary initial policy π
2. $\forall$ states $s \in S : \Pi(s) \leftarrow \{\pi(s)\}$
3. **repeat**
4. $Order \leftarrow$ Policy Evaluation(Π)
5. $\Pi \leftarrow$ Policy Improvement($Order$)
6. **until** Π stops changing

Figure 3.1.2: Qualitative Policy Iteration Algorithm

Lemma 3.1.4 suggests a qualitative policy iteration algorithm which only keeps track of the ordering of values of states, but not the values themselves, and, similarly, only requires the ordering of rewards as an input instead of the actual values. The pseudocode for the algorithm is given in Figure 3.1.2 . The key distinction between this algorithm and conventional policy iteration is that it works with pairs of states rather than single states. In each iteration, it updates the ordering between every pair of states based on new

ordering information received from the previous iteration. In doing so, it relies on the domain oracle which, given an ordering of states $s' \in \cup_{i=1}^{2} Next(s_i, a_i)$, returns '$<$' if $P(s'|s_1, a_1)$ strictly stochastically dominates $P(s'|s_2, a_2)$ with respect to the given ordering and '$>$' if $P(s'|s_2, a_2)$ strictly stochastically dominates $P(s'|s_1, a_1)$. The domain oracle can also indicate that neither [state, action] pair stochastically dominates the other one (or that it lacks knowledge to indicate dominance) by returning the unknown indicator '?'. If $P(s'|s_2, a_2)$ and $P(s'|s_1, a_1)$ stochastically dominate each other, the oracle returns '='. Similarly, the reward oracle returns '$< (>, =)$' if $r(s_1, a_1) < (>, =) r(s_2, a_2)$. While a domain oracle may seem hard to construct, we will demonstrate such oracles for two realistic control problems in Subsection 3.1.5. An example of a domain oracle in action is shown in Figure 3.1.3. It shows a car ascending a mountain, in two positions, one higher and one lower, moving with the same velocity. A possible ordering of next states appears as well, with states higher up the mountain being more valuable. With respect to this ordering, the car in the lower position on the mountain s_{low} has less of a chance of reaching more valuable states than the car in s_{high} and, therefore, $P_\pi(s'|s_{high})$ stochastically dominates $P_\pi(s'|s_{low})$ for any policy π.

Qualitative policy iteration is analogous to conventional policy iteration, with Order replacing the value function, and a set of deterministic candidate policies Π playing the role of the optimal policy. $\Pi : S \rightarrow 2^A$ is represented as a mapping from states to sets of actions, with each action $a \in \Pi(s)$ being possibly optimal in some quantitative instantiation of the qualitative MDP. In each iteration j, $Step\_Order^j$ represents the ordering on $V_{\pi,j}$, the expected rewards the agent will receive after j time steps in each state. In the example above, this ordering is updated after a call to the domain oracle to indicate that state s_{high} is more valuable than s_{low}. $Order$ instantiates the myopic comparison of the values of states - if neither one of the states s_{high} or s_{low} received any rewards prior to iteration j, then $Order$ is updated to indicate that s_{high} is more valuable.

The following theorem states that, when the qualitative policy iteration algorithm terminates, the optimal policy for any quantitative MDP consistent with the qualitative domain theory is contained in the returned candidate set of policies Π:

Theorem 3.1.5. *If qualitative policy iteration is executed in parallel with myopic policy iteration on any quantitative MDP consistent with the domain theory, at the end of iteration t, the candidate policy set Π contains the policy returned by the conventional policy iteration algorithm at the end of t.*

Proof. (sketch, see Theorem B.2.3 in the Appendix for full proof) For a fixed policy π, the ordering of values Order at iteration i of qualitative policy evaluation corresponds to the ordering of values of Myopic policy iteration at i (according to $\prec$). This can be seen by induction, with the base case given by the ordering of rewards, and the inductive step implied by Lemma 3.1.4 and the fact that, for any nonnegative

monotonically increasing function $V(s')$ with respect to order O on s', $\sum_{s'} P_1(s')V(s') < \sum_{s'} P_2(s')V(s')$ if P_2 strictly stochastically dominates P_1 with respect to O. This last fact (the proof of which can be found in Proposition B.2.1 in the Appendix) also implies that qualitative policy improvement does not eliminate the policy chosen by conventional policy iteration. □

Thus, the set of candidate policies returned by the algorithm on termination is guaranteed to contain the optimal policy. It is also possible to prove that this set of policies is independent of the choice of the initial policy, and that the running time of qualitative policy iteration is polynomial in the size of the state/action space ($O(|S|^4|A|^2)$, where $|S|$ is the total number of states and $|A|$ is the total number of actions). See discussion in Section B.3 of the Appendix for a proof of these facts.

3.1.4 Qualitative RL

When the set of returned policies is too large to be useful for the states of interest, an alternative is to combine qualitative specification of the problem with quantitative probability estimation. This is possible because stochastic dominance constraints have a precise probabilistic interpretation which can be useful for transferring knowledge between states via estimated probabilities. Consider the mountain car example in Figure 1. A priori, we may have reason to believe that all the states in S' are reachable from either s_{low} or s_{high} because the uncertainty in the power range is large enough to allow all of these transitions. Supppose however, that the agent discovers through experimentation that the highest state in S' is not reachable from s_{high}. Since we know that $P(s'|s_{high})$ stochastically dominates $P(s'|s_{low})$, it follows directly from the definition of stochastic dominance that it is not possible to reach that same highest state from s_{low}. Thus, the probability distribution of this, previously unseen transition can be updated with this new piece of knowledge. When performing qualitative reinforcement learning, we make the assumption that the domain oracle is correct, but the transition structure specified by the $Next$ mapping may be overly conservative (i.e., $Next(s, a)$ specifies a superset of states to which the agent can transition with nonzero probability upon executing $[s, a]$). This could happen, for example, if the domain expert overestimates the amount of noise affecting the output of the car's engine in the mountain car problem or the cart-pole problem described in the Introduction. Hence, the agent's task is to narrow down the transition structure. After each exploration episode, the agent performs the following sequence of steps:

1. Compute probability estimates $\widehat{P}(s'|s, a)$ for all $[s, a]$ pairs encountered during exploration based on collected statistics.

2. Infer values of $\widehat{P}(s'|s, a)$ for $[s, a]$ pairs, including those not encountered during exploration, based on

PROPAGATE($order, s_1, a_1, s_2, a_2$)
Input: $order \in \{'<','>','=','?'\}$, States $s_1 \in S, s_2 \in S$, Actions $a_1 \in A, a_2 \in A$
1. **switch** order
2. **case** '<':
3. $Y \leftarrow \{y \in S : \widehat{P}(\overline{Order}(y)|s_2, a_2) = 0\}$
4. $i \leftarrow 1$
5. **case** '>':
6. $Y \leftarrow \{y \in S : \widehat{P}(\underline{Order}(y)|s_1, a_1)) = 0\}$
7. $i \leftarrow 2$
8. **default** : $Y \leftarrow \varnothing$
9. **for all** $y \in Y$
10. **do** $\widehat{P}(y|s_i, a_i) \leftarrow 0$
11. $Next(s_i, a_i) \leftarrow Next(s_i, a_i) \backslash \{y\}$

QUAL ESTIMATION($Observed, Order$)
Input: Set of *Observed* transitions $[s, a]$, *Order* on S
1. **for all** pairs of states $s_1, s_2 \in S \times S$
2. **do for all** pairs of actions $a_1, a_2 \in \Pi(s_1) \times \Pi(s_2)$
3. **do** $order \leftarrow$ORACLE($Order$, s_1, a_1, s_2, a_2)
4. **if** $[s_2, a_2] \in Observed \wedge [s_1, a_1] \notin Observed$
5. **then** PROPAGATE($order, s_1, a_1, s_2, a_2$)

Figure 3.1.3: Qualitative Estimation Procedure

stochastic dominance constraints.

3. For all states s' such that $\widehat{P}(s'|s, a) = 0$ for some $[s, a]$, remove s' from $Next(s, a)$.

4. Perform qualitative policy iteration to determine the best policy for the updated Qualitative MDP.

In particular, the following proposition summarizes the inferences which can be made about unknown transition probabilities based on stochastic dominance constraints in step 2:

Proposition 3.1.6. *Let $P_1(s)$ stochastically dominate $P_2(s)$ with respect to some order O. Then if, for some y, $P_1(\overline{O}(y)) = 0$, then $P_2(y) = 0$. If, for some y, $P_2(\underline{O}(y)) = 0$, then $P_1(y) = 0$.*

Proof. The first statement follows immediately from Definition 3.1.3. The second statement follows from the first and the fact that if $P_1(s)$ stochastically dominates $P_2(s)$ with respect to O, then reversing the order O results in $P_2(s)$ stochastically dominating $P_1(s)$ with respect to reversed O . □

The full estimation algorithm is presented in Figure 3.1.3. It performs probability estimation based on Proposition 3.1.6. This algorithm is interleaved with the steps of the qualitative policy evaluation algorithm in Figure 3.1.2. After each iteration of policy evaluation, the definition of Next$(s, \pi(s))$ is updated to exclude states which have estimated zero transition probabilities.

3.1.5 Experiments

Two well-known domains were used to test the qualitative MDP algorithm: mountain car ascent and cart pole balancing. In the mountain-car task, the problem is to drive a car up a steep mountain (see Figure 3.1.3). The optimal policy depends on the power of the car's engine. If the engine is powerful enough to overcome gravity and drive the car up the slope to its goal, the optimal policy is to move towards the goal. Otherwise, the optimal policy is to move away from the goal up the opposite slope, and then apply full throttle to move towards the goal with the help of built-up inertia[2]. The agent received a reward of 1 upon reaching the goal. The task in the cart-pole problem is to balance a pole on a moving cart[3]. The reward of winding up in state s' was set to $cos\theta_t(s')$ to encourage actions which keep the pole as upright as possible. In both problems, actions which moved the agent out of bounds of the state space were disallowed.

The mountain car problem exhibits delayed rewards, while in the cart-pole problem rewards are immediate. Both of these problems represent a physical system which keeps track of its continuous state $z^t = [z_1^t, .., z_n^t]$ at time t through the update equations $z^{t+1} = [T_1(F; z^t), .., T_k(F; z_1^t), T_{k+1}(z^t), .., T_n(z^t)]$ which define the system's behavior under the influence of the input force F. In the simulation, the state space is discretized, with the boundary of the grid cell for each discrete state s given by $[\underline{s_i}, \overline{s_i}], i \in \{1, .., n\}$. The dynamics of the system are simulated by picking a characteristic position $z = Z(s)$ in each grid cell s and simulating each action from this position.

Qualitative MDP can be used to capture the situation when the engine's power is corrupted by unknown, but bounded noise. The power is modeled by a stochastic variable, distributed according to some unknown probability density function (pdf) with known support interval $[I_1, I_2]$ on which the pdf is strictly positive. This scenario uses a more realistic model of noise than the usual Gaussian noise assumption. It captures an infinite set of possible quantitative worlds, in each of which the noise distribution is known and the power pdf is completely specified on its support interval. We will show how to construct the domain oracle automatically under the assumption that the dynamics of the system are specified by invertible functions of the input force F, $T_i(F; z^t), i = 1, .., k$ and constant functions of F $T_i(z^t), i = k+1, .., n$. Both the mountain car and the cart-pole dynamics can be expressed in terms of such functions (see Section B.4 of

[2]The dynamics of car motion in terms of its position x_t and velocity $\dot{x}_t$ are given by the following equations:
$x_{t+1} = x_t + \dot{x}_{t+1}$, $\dot{x}_{t+1} = \dot{x}_t + Fa_t - Gcos(3x_t)$, where F is the amount of force applied by the engine, G is the pull of gravity, and $a_t \in \{+1$ (full throttle forward), -1 (full throttle reverse), and 0 (zero throttle)$\}$ is the action. We used $G = 1$ in our experiments. The state space was discretized by a 11×21 grid in the bounds $-1.2 \leq x \leq 0.5$, $-0.07 \leq \dot{x} \leq 0.07$.

[3]The state in the cart-pole problem is described by the angle between the pole and the vertical θ_t, the velocity of the cart $\dot{h_t}$, and the velocity of the pole $\dot{\theta}_t$. The update equations are:
$\ddot{\theta}_t = \frac{g*sin\theta_t + cos\theta_t[-Fa_t - m_p l\dot{\theta}_t^2 * sin\theta_t]/(m_c+m_p)}{l[4/3 - m_p * cos^2(\theta_t)/(m_c+m_p)]}$, $\ddot{h}_t = \frac{Fa_t + m_p l[\dot{\theta}_t^2 sin\theta_t - \ddot{\theta}_t cos\theta_t]}{m_c+m_p}$, $\dot{h}_{t+1} = h_t + \tau\ddot{h}_t$, $\theta_{t+1} = \theta_t + \tau\dot{\theta}_t$, $\dot{\theta}_{t+1} = \dot{\theta} + \tau\ddot{\theta}$, with gravity $g = 9.8$, cart mass $m_c = 1$, pole mass $m_p = 0.1$, distance from center of mass of the pole to the pivot $l = 0.5$, time step $\tau = 0.02$, and F is the force, $a_t \in -1, 1$ is the action. The state space was discretized into an $8 \times 8 \times 8$ grid in the range $-1.15 \leq \dot{h} \leq 1.15$, $-0.21 \leq \theta \leq 0.21$, $-2 \leq \dot{\theta} \leq 2$.

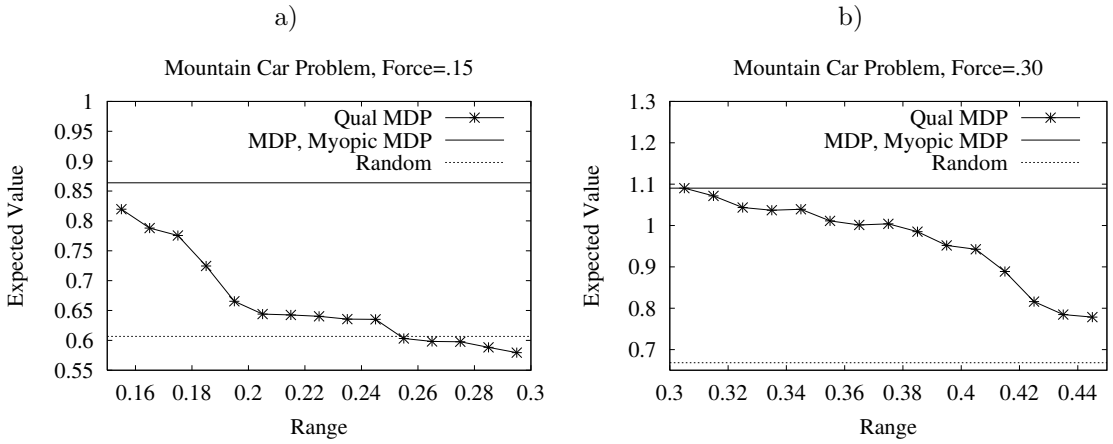


Figure 3.1.4: Performance of Qualitative Policies for the Mountain-Car Task. Plot of expected value of the policy versus the upper bound I_2 of the power support.

the Appendix).

For any $[s, a, s']$ tuple, we can determine the range of forces $\Phi(s'|s, a)$ under which applying action a in state s transitions the system to s' by inverting the transition dynamics as follows: $\Phi(s'|s, a) = \cap_{i=1}^{k}[T_i^{-1}(\underline{s'_i}; Z(s)), T_i^{-1}(\overline{s'_i}; Z(s))] \cap [I_1, I_2]$ if $\forall i \in \{k+1, .., n\}, T_i(Z(s)) \in [\underline{s'_i}, \overline{s'_i}]$ (assuming monotonically increasing $T_i(F; z^t)$, bounds are reversed for monotonically decreasing $T_i(F; z^t)$). $T_i^{-1}(z_i^{t+1}; z^t)$ is an inverse with respect to F. For a special case of linear functions $z_i^{t+1} = Q_i(z^t) + U_i(z^t)F$, the inverse is given by $T_i^{-1}(z_i^{t+1}; z^t) = \frac{z_i^{t+1} - Q_i(z^t)}{U_i(z^t)}$, assuming $U_i(z^t) \neq 0$ for any z^t. This special case applies in the mountain car and the cart-pole problems (see Section B.4 of the Appendix).

In order to handle sets of states S', we define $\Phi(S'|s, a) = \cup_{s' \in S'}\Phi(s'|s, a)$ as the set of forces under which applying action a in s transitions the system to one of the states $s' \in S'$. The next proposition states that stochastic dominance of probability distributions can be determined by checking the subset relationship for ranges of forces:

Proposition 3.1.7. *Let $\overline{O}(y) \cap Next(s, a)$ denote the set of next states for the transition $[s, a]$ which are at least as good as y with respect to order O. If $\forall y \in S, \Phi(\overline{O}(y) \cap Next(s_1, a_1)|s_1, a_1) \subseteq \Phi(\overline{O}(y) \cap Next(s_2, a_2)|s_2, a_2)$, then $P(s'|s_2, a_2)$ stochastically dominates $P(s'|s_1, a_1)$. Strict stochastic dominance holds when the subset is proper for some y.*

Proof. By property of probability, $A \subseteq B \Rightarrow P(A) \leq P(B)$, and $A \subset B \subseteq [I_1, I_2] \Rightarrow P(A) < P(B)$ by strict positivity of $P(x)$ on its support interval. □

In the first set of experiments, qualitative policy iteration was applied to a mountain car problem with the set of possible next state transitions constructed based on the power support interval $[0.15 - r, 0.15 + r]$

around the force $F = 0.15$. This power is insufficient to overcome the force of gravity, so the optimal policy has to move the car up the opposite slope first. Results are presented in Figure 3.1.4-(a), which compares the values of the random policy, the optimal policy for $F = 0.15$ for the conventional discounted MDP (with the discount factor $\alpha = 0.9$), the optimal policy for the same F for the Myopic MDP, and the set of policies Π computed via qualitative policy iteration. The expected value of each evaluated policy π is measured by the expected discounted return (with discount factor $\alpha = 0.9$) an agent would receive by randomly selecting a starting state s and following π thereafter. Set of policies Π was evaluated by choosing actions uniformly at random from $\Pi(s)$ in each state s. Qualitative policy iteration was evaluated on the increasing range r of the support interval. The policy degrades with increasingly uncertain power as Π becomes very large. For a wide range of power support intervals, the qualitative policy performs much better than random and, as the uncertainty interval decreases, its performance approaches that of the optimal MDP policy. Notice that for large power support sets, it is possible for qualitative policy (which is correct in some states and random in others) to be outperformed by the completely random policy. A similar experiment was repeated with $F = 0.3$, with similar results shown in Figure 3.1.4-(b). This force is large enough to overcome gravity and move the car directly to the goal from the bottom of the valley. The results of qualitative policy iteration for the pole-balancing task based on the increasing uncertainty range around $F = 35$ shown in Figure 3.1.6 are similar to the mountain car experiment. In the mountain car problem, the Myopic MDP performed as well as conventional MDP, and in the cart-pole problem, the performance gap (due to a more complex reward structure) was negligible.

An experiment was also performed to determine the sensitivity of the optimal policy to noise. The support interval for F was set to $[I_1, 0.35]$, with the lower bound I_1 starting at 0.33 and gradually decreasing to observe the degradation in certainty in the optimal policy in different states. The plot of the highest value I_1 at which the set of candidate policies returned by qualitative policy iteration contained more than one action (not counting equal-value actions) for that state is shown in Figure 3.1.5. The optimal policy for the states in the plateau regions never becomes uncertain because in those states, only one action is valid. A more interesting effect is the decreasing ridge of the uncertainty function - it shows that, as the amount of noise in F increases, the policy in states closest to the goal (but with car moving away from the goal) become uncertain first. States farther away from the goal (i.e., on the opposite slope) have enough potential energy to reach the goal with the forward throttle, even if the power is low, so the policy in those states stays certain longer. This experiment demonstrates the potential of qualitative policy iteration as a debugging tool of the MDP specification - it can allow the designer to determine the robustness of the optimal policy to noise in each region of the state space.

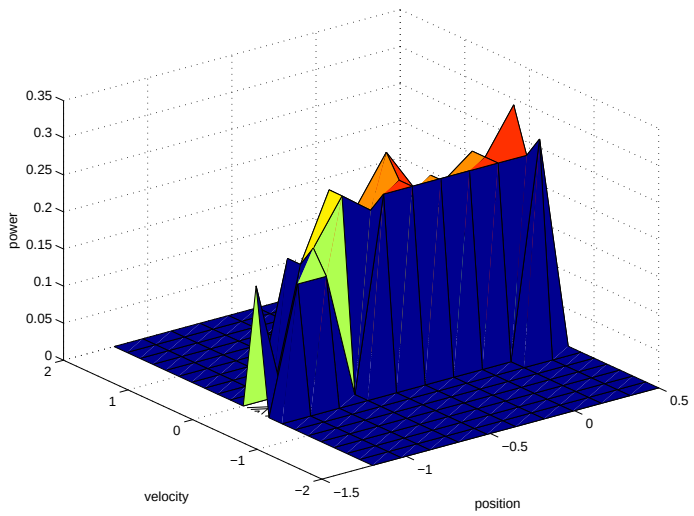


Figure 3.1.5: Sensitivity of Qualitative Policy as a function of states.

Finally, we experimented with qualitative reinforcement learning on the mountain car problem. A-priori, the engine power was specified with the uncertainty interval $F \in [0.15, 0.3]$ - too large to determine whether moving towards the goal or away from it is optimal on the bottom of the valley. The actual simulated process, however, applied forces to the throttle chosen uniformly from the interval $[0.15, 0.16]$ in each state (for which moving away from the goal was optimal). During each experimental episode, the car was always started at the bottom of the valley. Each episode terminated either when it reached the goal state or a state with no valid actions. The agent started out with $Next(s, a)$ set to all the states reachable when action/state pairs $[s, a]$ were encountered under the specified set of forces F. Whenever $P(s'|s, a)$ was estimated to be zero (either directly or through Proposition 3.1.6), s' was removed from the set $Next(s, a)$, prompting the domain oracle to disregard the forces $\Phi(s'|s, a)$ which could result in a transition to s' when action a was executed in s and accordingly update $F \leftarrow F \backslash \Phi(s, a)$. Then, all the possible next state sets $Next(s, a)$ for all the state/action pairs were recomputed based on the new F. Thus, QRL's generalization ability is due to the domain oracle performing a form of estimation of the power interval. This estimation allowed the oracle to eliminate possibilities that are not supported by empirical evidence.

Figure 3.1.7 shows the expected value of the optimal policy for both qualitative reinforcement learning and conventional model-based RL which applied policy iteration to estimated transition probabilities (actions in unencountered states were picked randomly). Both are shown as a function of the number of training episodes. Note that the episodes started out in the same state, but the performance metric averages over the random choice of an initial state. This metric reflects the algorithm's ability to generalize to unseen states. Since some states were not reachable from the starting state due to discretization of time and space,

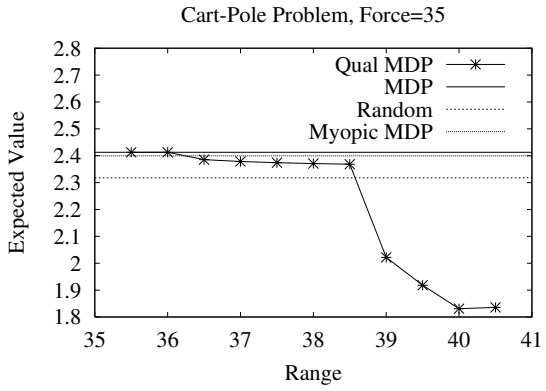


Figure 3.1.6: Performance of Qualitative Policies for Pole-Balancing Task.Plot of averaged expected value of the policy versus the upper bound I_2 of the power support.

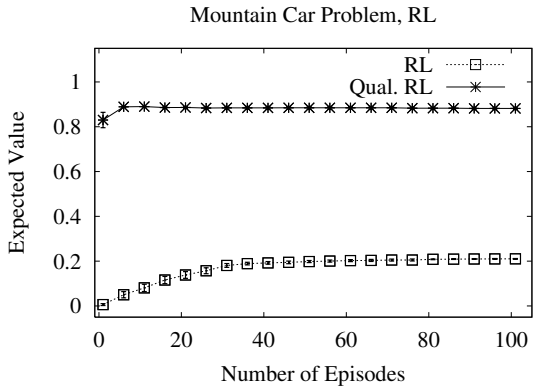


Figure 3.1.7: Performance of Qualitative and Conventional RL on the mountain car task. The performance is displayed with 95% error bars based on 100 different episodes for conventional reinforcement learning and 10 different episodes for qualitative reinforcement learning.

RL never learned how to act in them, resulting in poor performance. QRL outperformed RL by deducing how to act in unseen states based on information propagated from encountered states.

3.1.6 Conclusion

In many MDP problems, it is desirable to avoid exploration as an expensive and potentially dangerous process. We presented an algorithm which either eliminates the need to explore the environment altogether while requiring a much less precise description of the problem than an MDP, or limits the amount of exploration needed to achieve the optimal policy.

3.2 Active Reinforcement Learning

3.2.1 Introduction

When the transition probabilities and rewards of an MDP are known, the optimal policy can be computed offline. However, it is unrealistic to expect a domain expert to accurately specify numeric MDP parameters. The optimal policy computed offline in an imperfectly modeled world, therefore, may turn out to be suboptimal when executed in the actual environment. To fix this problem in practice, both rewards and transition probabilities are tweaked by domain experts until the desired performance is achieved. An alternative approach is to allow the agent to explore the world in a model-free fashion using reinforcement learning (RL). However, reinforcement learning in an actual environment is time-consuming, expensive, and sometimes dangerous ([4], for example, describe a helicopter crash which occurred during overly aggressive exploration).

In this section, we introduce an approach called active reinforcement learning which combines the strengths of offline planning and online exploration. In particular, our framework allows domain experts to specify possibly inaccurate models of the world offline. However, instead of using this model for planning, our algorithm uses it as a blueprint for exploration. Our approach is based on the observation that, while all of the transition probabilities and rewards in the model may be misspecified, it is not important to know all of them to determine the optimal policy. Consider, for example, a surveillance helicopter flying agent. Does it make a difference if it crashes with probability .9 or .95 when it flies close to the ground? It seems unlikely that the optimal policy would be very sensitive to this value. However, the probability of the agent taking a good photograph of its target from a given viewing angle is extremely important. Therefore, the primary goal of the agent's experimentation, given a description of the problem, should be to determine the probabilities of capturing a photo of the target as opposed to trying to determine the exact probability of crashing. Active reinforcement learning enables this type of exploration. It uses sensitivity analysis to determine how the optimal policy in the expert-specified MDP is affected by changes in transition probabilities and rewards of individual actions. This analysis guides the exploration process by forcing the agent to sample the most sensitive actions first. We will present experimental results demonstrating the effectiveness of active RL. In addition, we will show that, while our algorithm is approximate, it produces near-optimal results in polynomial time for a special class of MDPs.

3.2.2 Related Work

Many strategies have been proposed to address the difficulty of specifying MDPs offline. Givan's bounded-parameter MDP framework [39] allows the designer to specify a set of possible worlds via uncertainty intervals around MDP's transition probabilities and rewards. The agent then finds the best policy in a game against adversarial nature which, given the agent's policy, picks the worst possible world in which to evaluate it. While this approach is useful, it may pick an overly conservative policy which is far from optimal for a given environment.

An approach similar in spirit to ours is Bayesian reinforcement learning [24], which imposes a prior distribution over possible worlds and updates it based on interactions with the environment. This prior can be used to capture expert's domain knowledge. However, this approach requires making unrealistic assumptions on the shapes of probability distribution functions to ensure tractability. Even with these assumptions, it resorts to sampling to approximate probabilities. The largest problem to which it was applied has fifty six states. The problems we solve in this work are two orders of magnitude larger. In addition, we present an approximate version of our algorithm which is able to handle much larger (possibly continuous) state/action spaces.

The idea of using the MDP specification to reduce the sample complexity of RL has also been explored by [5]. However, they only deal with a deterministic environment. More importantly, their exploration is driven by the perceived optimal policy, not sensitivity analysis.

3.2.3 Active RL Algorithm

In this section, we define our notation and give a general overview of the active reinforcement learning algorithm.

The Markov decision process is defined by a tuple $(S, A, T = \overline{T} \cup \underline{T}, Next = \overline{Next} \cup \underline{Next}, R, \alpha)$, where S is a finite set of states, A is a finite set of actions, $R(s,a)$ is a reward function, $T(s'|s,a)$ is a transition probability function, $\alpha \in (0,1)$ is the discount factor. $Next(s,a)$ defines a set of states reachable in one step with nonzero probability after taking action $a \in A$ in a state $s \in S$. $\overline{Next}(s,a)$ is a set of states with independent transition probabilities $\overline{T}(s'|s,a)$, $s' \in \overline{Next}(s,a)$, while $\underline{Next}(s,a)$ defines a single state such that $\underline{T}(\underline{Next}(s,a)|s,a) = 1 - \sum_{s' \in \overline{Next}(s,a)} \overline{T}(s'|s,a)$. Let $\pi(s)$ define a deterministic policy which maps states to actions. The utility of this policy $U^{\pi}_{T,R}$ is given by the expected discounted rewards:

$$U^{\pi}_{T,R} = E_{s_0 \sim D; s_{t+1} \sim T(\cdot|s_t, \pi(s_t))}[\sum_{t=1}^{\infty} \alpha^t R(s_t, \pi(s_t)))|\pi, T, R],$$

with initial state s_0 drawn from the distribution D. The utility of a policy explicitly depends on the transition and reward model of the MDP. We can use Taylor's approximation to model the local sensitivity of $U^{\pi}_{T_0,R_0}$ as the transition probabilities $\overline{T}(\cdot|s,a)$ are perturbed around the specified value $\overline{T_1}(\cdot|s,a)$ for a single state/action pair s,a:

$$\hat{U}^{\pi}_{T_1(\cdot|s,a)}(\overline{T}(\cdot|s,a)) \approx U^{\pi}(\overline{T}_1(\cdot|s,a)) + \nabla_{\overline{T}(\cdot|s,a)} U^{\pi}(\overline{T}_1(\cdot|s,a))(\overline{T}(\cdot|s,a) - \overline{T_1}(\cdot|s,a))^T$$

All the transition probabilities other than those of action a in state s are held fixed at the values defined by the user-supplied model T_0, R_0. The sensitivity of the utility function to changes in individual rewards can be modeled in an analogous fashion.

Taylor's approximation makes it possible to determine how the payoff from following a fixed policy is affected by the changes in the MDP parameters. However, even large changes in payoffs do not necessarily mean that the agent is acting suboptimally. An extreme illustration of this phenomenon is a gridworld agent who is rewarded only upon getting to the goal state. Even if the agent is wrong about the magnitude of the reward, its optimal policy remains the same: always move towards the goal. Thus, an agent could be completely wrong about the environment and still act optimally. The key goal of the sensitivity analysis is to determine how the *optimal policy* changes in response to the changes in the transition probabilities and rewards. One way to measure this sensitivity is by asking the question: how much do transition probabilities/rewards of a given action have to change before the currently optimal policy becomes suboptimal?

To make this question precise, let us first focus on the sensitivity to transition probabilities. We will use $\Pi_{T_1(\cdot|s,a)} = \arg\max_{\pi} U^{\pi}_{T_1(\cdot|s,a),R_0}(\overline{T}_1(\cdot|s,a))$ to denote the optimal policy in the MDP in which all transitions except for those of action a are held fixed at T_0, and transitions of a are given by $T_1(\cdot|s,a)$. Let $C = \{T(\cdot|s,a) : U^{\Pi_{T_0(\cdot|s,a)}}(\overline{T}(\cdot|s,a)) \geq U^{\Pi_{T(\cdot|s,a)}}(\overline{T}(\cdot|s,a))\}$ define a region in the transition probability space in which the optimal policy $\Pi_{T_0(\cdot|s,a)}$ for the user-specified MDP dominates every other policy. The goal of sensitivity analysis is to find the radius of the largest ball which we can position at T_0 and expand without leaving the confines of C along the dimensions $\overline{T}(\cdot|s,a)$ corresponding to the transitions for a given action a. The larger the ball, the more robust the optimal policy is to the changes in the transition probabilities of the given action a. However, it is not obvious how to compute this quantity exactly. Instead, we approximate it via a variant of Newton's root-finding method which starts out at some point T_0' in the transition probability space outside of C and converges to a point on the boundary of C. The method is illustrated in Figure 3.2.1. Each application of the method consists of starting out at some point T_0', replacing the utility function $U^{\Pi_{T_0'(\cdot|s,a)}}(T_0'(\cdot|s,a))$ of the best policy at T_0' with its tangent, finding the "zero" of this tangent, i.e., a point

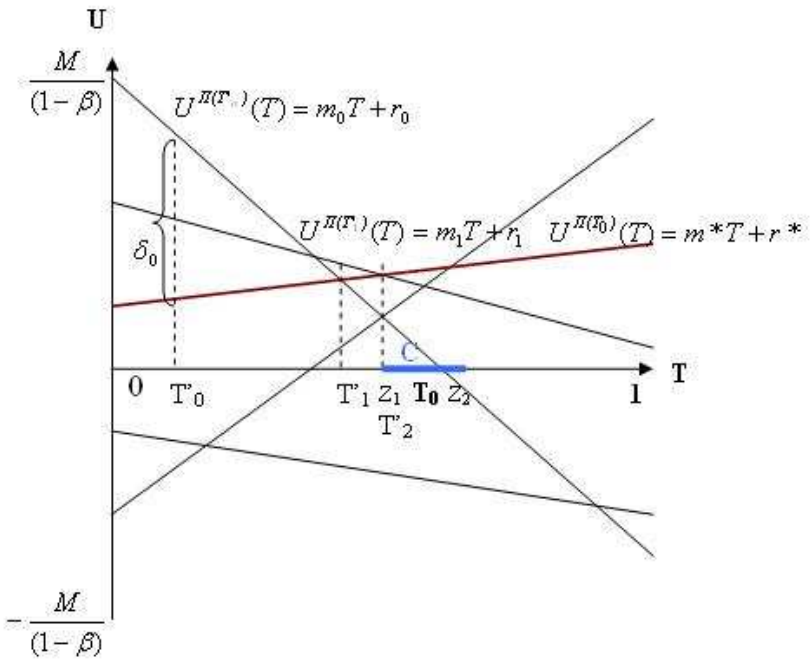


Figure 3.2.1: Newton's method for sensitivity analysis of MDPs. The utilities of individual policies are linear functions of T. Function $U^{\Pi_{T(\cdot|s,a)}}(\overline{T}(\cdot|s,a))$ is the utility of the optimal policy at $T(\cdot|s,a)$ as a function of $\overline{T}(\cdot|s,a)$. It is given by the upper envelope of the set of all value functions. C is the region where the optimal policy at T_0 (represented by a red line if color is available) is dominant. The algorithm starts out at T'_0 and converges to T'_2 on the boundary of C.

T'_1 closest to T_0 in the intersection of this tangent and the tangent to the utility function at T_0, replacing the initial estimate T'_0 with the new estimate T'_1, and iterating. The algorithm is repeated with several choices of the starting point, and the minimum of the distances between each final iterate and T_0 is used as an approximate radius of the largest ball enclosed in C.

The pseudocode for this algorithm consists of the following steps:

1. Determine the optimal policy $\Pi_{T_0(\cdot|s,a)}$ for the user-provided model using any MDP solver.

2. Determine the Taylor approximation of the utility of this policy $\hat{U}^{\Pi_{T_0(\cdot|s,a)}}_{T_0(\cdot|s,a)}(\overline{T}(\cdot|s,a))$ as a function of transition probabilities $\overline{T}(\cdot|s,a)$.

3. Select the starting point $T'_0(\cdot|s,a)$ and let $i \leftarrow 0$.

4. Using any MDP solver, determine the optimal policy $\Pi_{T'_i(\cdot|s,a)}$ for the user-provided model with transition probabilities of action a replaced by $T'_i(\cdot|s,a)$.

5. Determine the Taylor approximation of the utility of this policy $\hat{U}^{\Pi_{T_i'(\cdot|s,a)}}_{T_i'(\cdot|s,a)}(\overline{T}(\cdot|s,a))$ as a function of transition probabilities $\overline{T}(\cdot|s,a)$.

6. Let $T'_{i+1}(\cdot|s,a)$ be the point in the intersection of $\hat{U}^{\Pi_{T_i'(\cdot|s,a)}}_{T_i'(\cdot|s,a)}(\overline{T}(\cdot|s,a))$ and $\hat{U}^{\Pi_{T_0(\cdot|s,a)}}_{T_0(\cdot|s,a)}(\overline{T}(\cdot|s,a))$ closest to the user-specified model T_0. This is done by solving the following second-order cone program[4]:

$$\begin{aligned}\overline{T'_{i+1}}(\cdot|s,a) = \arg\min_X \left\|X - \overline{T_0}(\cdot|s,a)\right\| \\ \text{s.t.} \hat{U}^{\Pi_{T_i'(\cdot|s,a)}}_{T_i'(\cdot|s,a)}(X) = \hat{U}^{\Pi_{T_0(\cdot|s,a)}}_{T_0(\cdot|s,a)}(X) \\ X \succeq \mathbf{0}; X^T \boldsymbol{e} \leq 1, \boldsymbol{e} \text{ is a vector of all 1's}\end{aligned}$$

   The last set of constraints ensures that $T'_{i+1}(\cdot|s,a)$ is a valid probability distribution.

7. Let $i \leftarrow i+1$ and repeat steps 4-7 while $\Pi_{T_i(\cdot|s,a)} \neq \Pi_{T_0(\cdot|s,a)}$.

8. Return $\|T'_i(\cdot|s,a) - T_0(\cdot|s,a)\|$

The algorithm converges to a boundary point of C. It is known that Newton's method is sensitive to the choice of the initial point T'_0. Therefore, we executed our experiments with $T'_0(\cdot|s,a)$ initialized to each vertex of the probability simplex of $T(\cdot|s,a)$. The distance of the closest final point to T_0 was then chosen to represent the sensitivity of the optimal policy to the given action.

An analogous algorithm is used to determine the sensitivity of the MDP to perturbations in individual rewards. We have not yet described how to compute the Taylor approximation of the utility function. We will do so in the next section.

3.2.4 Sensitivity of a Policy

Let $P^\pi(s'|s) = T(s'|s,\pi(s))$ be the transition probability function under π. For given T_0, R_0, and a fixed policy π, the value function $V_0^{\pi,T_0,R_0}(s)$ under π is given by the Bellman equation

$$V_0^\pi = \alpha P_0^\pi V_0^\pi + R_0^\pi$$

V_0^π can be computed efficiently via iterative application of the Bellman equation, known as policy evaluation. The utility of a policy can be computed by $U^\pi_{T,R} = E_{s\sim D} V_0^{\pi,T,R}(s)$.

[4]Second-order cone programs (SOCPs) are a special case of semidefinite programs which can be solved more efficiently, see [51] for an overview.

By definition of the gradient and linearity of expectation, the gradient of the utility function is given by

$$\nabla_{\overline{T}(\cdot|s,a)} U^{\pi}(T_1(\cdot|s,a)) = \left[E_{s\sim D} \frac{\partial V^{\pi}(T_1(\cdot|s,a))}{\partial \overline{T}(s'_1|s,a)} \ldots E_{s\sim D} \frac{\partial V^{\pi}(T_1(\cdot|s,a)}{\partial \overline{T}(s'_{|\overline{Next}(s,a)|}|s,a)} \right].$$

In order to compute the gradient of the value function, we need the following lemma:

Lemma 3.2.1. *For a given policy π and a [state,action,next state] tuple $[s,a,s'] : s' \in \overline{Next(s,a)}$, $\frac{\partial V}{\partial \overline{T}(s'|s,a)} \triangleq 0$ for $a \neq \pi(s)$. For $a = \pi(s)$, $\frac{\partial V}{\partial \overline{T}(s'|s,a)} = \frac{\alpha}{\epsilon}(I - \alpha P_0^{\pi})^{-1} L V_0$, where*

$$L(s_j|s_i, \pi(s_i)) \triangleq \begin{cases} \epsilon, & \text{if } s_j = s' \\ -\epsilon, & \text{if } s_j \in \underline{Next}(s,a) \\ 0, & \text{otherwise} \end{cases}$$

Proof. The proof is a slight modification of the analysis given in [17]. For completeness, it is given in the Appendix, Theorem C.1.1. □

In order to compute the directional derivatives efficiently, note that the equation $\frac{\partial V}{\partial \overline{T}(s'|s,a)} = \frac{\alpha}{\epsilon}(I - \alpha P_0^{\pi})^{-1} L V_0$ can be rewritten as a recursion $\frac{\partial V}{\partial \overline{T}(s'|s,a)} = \alpha P_0^{\pi} \frac{\partial V}{\partial \overline{T}(s'|s,a)} + \frac{\alpha}{\epsilon} L V_0$. The form of this equation is exactly the same as that of the Bellman equation, with the value function replaced by its derivative and the reward function replaced by $\frac{\alpha}{\epsilon} L V_0$. Therefore, policy evaluation can be used to compute $\frac{\partial V}{\partial \overline{T}(s'|s,a)}$. A similar analysis applies when the value function is expressed as a function of reward for a given state/action pair $[s,a]$: $V(R(s,a)) = V_0 + \frac{\partial V}{\partial R(s,a)}(R(s,a) - R_0(s,a))$, with $\frac{\partial V}{\partial R(s,\pi(s))} = \alpha P_0 \frac{\partial V}{\partial R(s,\pi(s))} + D$ where $D(s_i, \pi(s_i)) \triangleq \begin{cases} 1, \text{ if } s_i = s \\ 0, \text{ otherwise} \end{cases}$ and $\frac{\partial V}{\partial R(s,a)} \triangleq 0$ for $a \neq \pi(s)$.

3.2.5 Convergence and Complexity

In this section, we consider issues connected with the algorithm's convergence and its complexity. The geometric structure of our algorithm is similar to policy iteration which has well-known connections to Newton's method [60, 52]. Just like in policy iteration, the known local quadratic convergence of Newton's method does not ensure global polynomial time complexity. Unlike policy iteration, we cannot rely on properties of contractions to establish convergence. However, we can establish convergence for MDPs whose structures (given by transitions with nonzero probabilities) are directed acyclic graphs (dags). This is because for such MDPs, the values $U^{\pi}(\overline{T}(\cdot|s,a)$ are linear functions of $\overline{T}(\cdot|s,a)$ for single state/action pairs $[s,a]$. We have the following result:

Theorem 3.2.2. *Suppose that for every T and every policy π, $U^\pi(T)$ is a linear function of T. Then, for any initial point T'_0, the sequence $\{T'_i\}$ generated by the variant of Newton's method in Subsection 3.2.3 converges to the boundary of C in a finite number of steps.*

Proof. (sketch, see Theorem C.1.2 in the Appendix for full proof) Since the optimal policy in iteration i of the algorithm $\Pi_{T'_i(\cdot|s,a)}$ is suboptimal when the transition probabilities of a are changed to $\overline{T_0}(\cdot|s,a)$ and vice versa, the optimal policy at $\overline{T_0}(\cdot|s,a)$ is suboptimal at $\overline{T'_i}(\cdot|s,a)$, by the Intermediate Value Theorem, there must be at least one point P along the line connecting $\overline{T'_i}$ and $\overline{T_0}$ where utility functions of these two policies intersect. Since P lies in the feasible region of the convex program in step 6 of the algorithm by convexity of the probability simplex, $\overline{T'_{i+1}}$ is no farther away from $\overline{T_0}$ than P, and the algorithm makes monotonic progress towards $\overline{T_0}$ in each iteration. Moreover, it never picks the same policy twice, which implies that it converges in a finite number of iterations. □

For dag-structured MDPs such that the maximum number of $Next$ states for any state/action pair is two, much stronger guarantees are available. As pointed out in [52], any MDP can be converted in polynomial time into an MDP of this form by introducing extra states and transitions as necessary. The significance of these MDPs is that, since $\overline{Next}(s,a)$ is a singleton, $U^\pi(\overline{T}(\cdot|s,a))$ is a linear function of a single variable $\overline{T}(\cdot|s,a)$. A binary search in this case can determine the radius of the region C in logarithmic time. The following theorem shows that our algorithm performs as well as the binary search in this degenerate case:

Theorem 3.2.3. *Define the distance between two policies π and π' as $max_{T(\cdot|s,a)}|U^\pi(\overline{T}(\cdot|s,a))-U^{\pi'}(\overline{T}(\cdot|s,a))|$. Let γ be the smallest distance larger than 0 between any policy and $\Pi_{T_0(\cdot|s,a)}$. Let the MDP have bounded rewards: $|R_0(s,a)| \leq M$ for $\forall s,a$. Then, after $t = O(\log(\frac{M}{\gamma\epsilon(1-\alpha)}))$ iterations, the iterates $T'_{i\geq t}$ of Newton's method are within ϵ of the limit point on the boundary of C.*

Proof. (sketch, see Proposition C.1.3 and Theorem C.1.6 in the Appendix for full proof) The proof follows from a known fact that one-dimensional Newton's method makes progress in each iteration by either exponentially decreasing the height or exponentially increasing the slope of the function $U^\pi(\overline{T}(\cdot|s,a))$ [52]. □

If the structure of an MDP is not a dag, then $U^\pi(\overline{T}(\cdot|s,a))$ is not linear in $\overline{T}(\cdot|s,a)$ and the above convergence results no longer apply to the Newton's method for determining sensitivity to transition probabilities. However, as the discount factor $\alpha \to 0$, the value function for any MDP will become approximately linear as the influence of distant rewards becomes negligible. Thus, Newton's method offers a way to find an approximate solution to our problem which becomes more accurate as the discount factor decreases. For general root-finding problems, Newton's method need not converge (i.e., it may cycle or diverge to infinity).

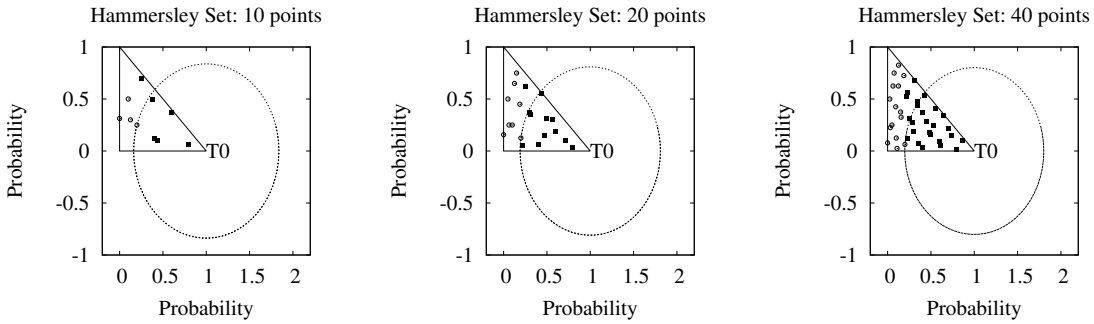


Figure 3.2.2: Hammersley Point Sets for Active RL. The triangle delineates the boundary of the probability simplex. Dark points are inside C, light points are outside. The sphere centered at T_0 excludes all of the points outside of C.

It is likely that our variant may also exhibit this undesirable behavior when applied to arbitrary MDPs. Our experimental results, however, demonstrate that the method works well in practice.

3.2.6 Optimal Active RL

In Subsection 3.2.3, we have presented an algorithm for determining the robustness of the optimal policy with respect to perturbations in transition probabilities of individual actions. The goal of the algorithm is to determine the radius of the largest ball that can be placed inside the region C in the transition probability space in which the optimal policy dominates every other policy. However, as noted in the previous section, our algorithm based on Newton's method gives an approximate answer only, except in the special cases noted in the previous section. In general, there are no guarantees on the quality of this approximation. In this section, we present an alternative algorithm with strong theoretical guarantees. The idea behind the algorithm is to fill up the transition probability simplex with points. Each point represents a particular perturbation in the action's transition probabilities. Policy iteration is then used to find the optimal policy for each perturbation, and determine when this optimal policy is different from the optimal policy $\Pi_{T_0(\cdot|s,a)}$ for the user-specified MDP T_0. The algorithm then places a sphere at the user-specified transition probabilities $T_0(\cdot|s,a)$ which only includes points in C, the region where $\Pi_{T_0(\cdot|s,a)}$ dominates every other policy (see Figure 3.2.2). The algorithm is given by the following pseudocode:

1. Generate a point set P in the probability simplex $\overline{T}(\cdot|s,a)$.
2. Let $X \subseteq P$ be the subset of points for which the policy $\Pi_{T_0(\cdot|s,a)}$ is suboptimal.
3. Return $\min_{\overline{T'} \in X} \left\| \overline{T'} - \overline{T_0}(\cdot|s,a) \right\|$.

A key question is how to fill up the probability simplex with points in step (1) of the algorithm. Since this is a sampling algorithm, we want to fill up the space inside the simplex as densely as possible to make sure that we do not miss the boundary of C. We will show that such "dense" point sets result in good estimates of the radius of the sphere inside C. In order to do so, we need the following definition of dispersion of a point set:

Definition 3.2.4. Dispersion of a point set P which consists of points $\{x_1, \ldots x_N\}$, in a closed set $V \subseteq \mathbb{R}^n$, is defined as $d(P) = sup_{x \in V} min_{1 \leq i \leq N} \|x - x_i\|$.

The dispersion of a point set P is given by the smallest radius R such that the balls $B(x_1, R) \ldots B(x_N, R)$ of radius R centered at the points $x_1, \ldots, x_N$ cover the set V. The following theorem shows that the point sets with low dispersion result in good estimates of the robustness of the optimal policy to changes in transition probabilities of actions, as measured by the radius of the largest sphere inscribed into C:

Theorem 3.2.5. *Let T define the probability simplex of any action a of an MDP with the discount factor α and rewards in the interval $[-M, M]$. Let P be a point set on the probability simplex T with dispersion $d(P) < \epsilon$, and let $B(T_0(\cdot|s,a), R)$ be a sphere centered at $T_0(\cdot|s,a)$ determined by the algorithm above (with R given by the return value in step (3)). Then, for any point $Z \in B(T_0(\cdot|s,a), R)$, the policy $\Pi_{T_0(\cdot|s,a)}$ is not too suboptimal in the world with transition probabilities of action a perturbed to Z. More formally, its utility is not much worse than the utility of any other policy π: $U^{\Pi_{T_0(\cdot|s,a)}}(Z) > U^{\pi}(Z) - 2K\epsilon$, where K is given by $\frac{2\alpha M}{(1-\alpha)^2}$.*

Hence, it is preferable to use point sets with low dispersion. Fortunately, a method of constructing such sets is known. One such construction is called a Hammerseley Point Set, and its dispersion may be bounded by the following known theorem [59]:

Theorem 3.2.6. *If $s \geq 2$, then, for the N-element Hammersley point set P in the probability simplex in $\mathbb{R}^s$ in the pairwise relatively prime bases $b_1, \ldots, b_{s-1}$ we have $d(P) < (1 + \max_{1 \leq i \leq s-1} b_i) N^{-1/s}$.*

Proof. The proof for Hammersley point sets on the unit cube (rather than the probability simplex) is given in [59]. We modified the Hammersley point set construction by constructing a Hammersley point set on the unit cube, and then reflecting the points in the upper simplex about the hyperplane $x_1 + \ldots + x_n = 1$ to produce a point set in the probability simplex. This construction does not decrease dispersion. □

This theorem justifies the use of Hammersley point sets for sensitivity analysis. Three such point sets in the probability simplex in $\mathbb{R}^2$ are shown in Figure 3.2.2, with the number of points N varying from 10 to 20 to 40. The corresponding sphere estimated via the optimal Active RL algorithm is also shown. As a

consequence of Theorems 3.2.6 and 3.2.5, the optimal Active RL algorithm will return the correct radius of the largest sphere inscribed into C as the number of points N increases to infinity. However, this algorithm is much slower than the Newton's method variant introduced in Subsection 3.2.3 and is not practical for higher-dimensional transition spaces.

3.2.7 Approximate Active RL

In worlds with very large or continuous state/action spaces, the exact Active RL algorithm of Subsection 3.2.3 is intractable. Moreover, in continuous state spaces, our notion of the sensitivity of individual state/action pairs no longer applies. Instead, we consider the case where the state space is partitioned into regions $\mathbb{B}$ with the uncertainty in our transition model for each region generated by a random variable. Formally, we assume that the world is governed by the control model $x_{t+1} = f(x_t, a, D)$. The agent's state x at time $t+1$ is a (possibly nonlinear) *function* f of the agent's state at time t, its action a, and the disturbance input D. The disturbance D is a random variable with distribution $T(\cdot|B^{x_t})$ which depends on the region B^{x_t} of the state space the agent is in. Analogously to the standard MDP, this distribution is defined on the domain $Next(B) = \{D'_1(B), .., D'_{|\overline{Next}(B)|}(B)\} \cup \underline{Next}(B)$. The approximate active RL procedure determines which regions $B \in \mathbb{B}$ affect the optimal policy the most.

Consider, for example, the sailing problem illustrated in Figure 3.2.3. The agent's goal in this domain is to get the sailboat to the finish line. The sailboat is controlled by the rudder and the sail. The problem is complicated by a probabilistic whirlpool which could aid the agent by increasing its speed or detain it by deviating the boat from its course. In this case, the function f describes the sailboat dynamics given the boat's position, the rudder/sail action, and the weather state D. This weather state is composed of a deterministic wind and a probabilistic current. The strength of the current and its direction are given by a probabilistic distribution T which depends on the band B inside the whirlpool in which the agent finds itself. Without knowing the direction of the current, it is impossible to determine which one of the policies shown in Figure 3.2.3 is optimal. However, it turns out that it is not necessary to know the current in all the bands in order to determine the optimal policy. The approximate active RL procedure determines the sensitivity of the optimal policy to each of the regions $B \in \mathbb{B}$.

The approximate active RL procedure is analogous to its exact variant, except that local policy search is used in place of an MDP solver, and the gradient of the utility function with respect to transition probabilities is approximated. Since we are interested in the gradient with respect to a region of the state space instead of an individual state, Lemma 3.2.1 no longer applies. Therefore, we approximate the gradient linearly by perturbing the transition probabilities by ϵ in each dimension of the probability simplex:

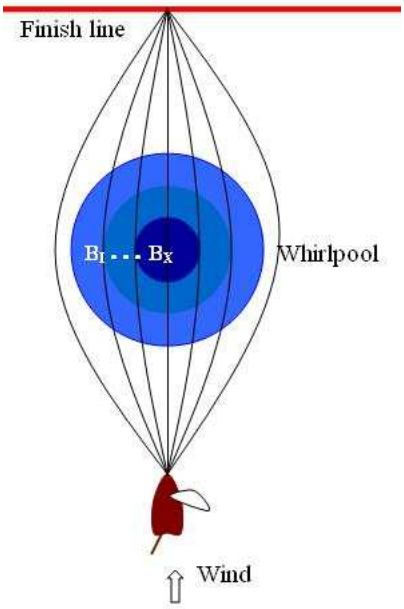


Figure 3.2.3: Sailboat Domain

$\nabla_{\overline{T}(\cdot|B)} U^{\pi}(\overline{T_1}(\cdot|B)) \approx \frac{1}{\epsilon}[\widetilde{U}^{\pi}_{K_1(\cdot|B)} - \widetilde{U}^{\pi}_{T_1(\cdot|B)}, \ldots, \widetilde{U}^{\pi}_{K_{|\overline{Next}(\cdot|B)|}(\cdot|B)} - \widetilde{U}^{\pi}_{T_1(\cdot|B)}]$, where $T_1(\cdot|B)$ denotes the world in which all of the transition probabilities in regions $B' \neq B$ are fixed at $T_0(\cdot|B')$ and the transitions in region B are fixed at $T_1(\cdot|B)$; $K_j(\cdot|B)$ denotes the same world, but with transition probability $T_1(D'_j(B)|B)$ perturbed by ϵ, and transition probability $T_1(\underline{Next}(B)|B)$ perturbed by $-\epsilon$. In each world W, the utility of the policy π (denoted by $\widetilde{U}^{\pi}_{W}$) is approximated by Markov Chain Monte Carlo (MCMC).

The following version of our active RL algorithm calculates the sensitivity of the optimal policy to the transition probabilities in each band $B \in \mathbb{B}$:

1. Determine the optimal policy $\Pi_{T_0(\cdot|B)}$ for the user-provided model using a local policy search[5]. In the sailboat example, local searches are conducted in several regions which correspond to going through the left part of the whirlpool, the right part, the center, or avoiding the whirlpool altogether (see Figure 3.2.3). The optimal policy is then determined by comparing the utilities of the optimal policies found by each local search. The utility of each policy is determined by MCMC.

2. Determine the Taylor approximation of the utility of this policy $\hat{U}^{\Pi_{T_0(\cdot|B)}}_{T_0(\cdot|B)}(\overline{T}(\cdot|B))$ as a function of transition probabilities $\overline{T}(\cdot|B)$ for the chosen region B.

3. Select the starting point $T'_0(\cdot|B)$ in the same manner as the exact case and let $i \leftarrow 0$.

[5]Any local policy search algorithm, such as policy gradient or dynamic programming, can be used for this procedure.

4. Using local policy search as described in step (1), determine the optimal policy $\Pi_{T_i'(\cdot|B)}$ for the user-provided model with transition probabilities replaced by $T_i'(\cdot|B)$.

5. Determine the Taylor approximation of the utility of this policy $\hat{U}_{T_i'(\cdot|B_1)}^{\Pi_{T_i'(\cdot|B)}}(\overline{T}(\cdot|B))$ as a function of transition probabilities $\overline{T}(\cdot|B)$.

6. Let $T_{i+1}'(\cdot|B)$ be the point in the intersection of $\hat{U}_{T_i'(\cdot|B)}^{\Pi_{T_i'(\cdot|B)}}(\overline{T}(\cdot|B))$ and $\hat{U}_{T_0(\cdot|B)}^{\Pi_{T_0(\cdot|B)}}(\overline{T}(\cdot|B))$ closest to the user-specified model T_0. This is done by solving a SOCP, analogous to step (6) of the exact RL algorithm.

7. Let $i \leftarrow i+1$. Use MCMC to compute $d(W) \leftarrow |\widetilde{U}_{W(\cdot|B)}^{\Pi_{T_i(\cdot|B)}} - \widetilde{U}_{W(\cdot|B)}^{\Pi_{T_0(\cdot|B)}}|$ in several different sampled worlds W. Repeat steps 4-7 while $\arg\max_W d(W) > \xi$.

8. Return $\|T_i'(\cdot|B) - T_0(\cdot|B)\|$

Besides the reliance on MCMC to approximate utilities and gradients, an important distinction between this algorithm and the exact variant is the terminating condition in step (7). Since policies in a continuous world can be arbitrarily similar to each other, we judge two policies to be approximately equal if their utilities are close enough to each other (no farther than a small constant ξ) as the transition probabilities $W(\cdot|B)$ are allowed to vary freely. Thus, the algorithm is estimating, for each region B, the largest perturbation in transition probabilities $T_0(\cdot|B)$ in which the optimal policy $\Pi_{T_0(\cdot|B)}$ is not too suboptimal. Experimental results for the sailing domain (as well as various problems solved with exact active RL) are presented in the next section.

3.2.8 Experiments

Exact RL experiments were performed on the following domains:

- In the mountain-car task, the problem is to drive a car up a steep mountain [70]. In our setup, the engine power was taken to be random, uniformly distributed in the interval $[.15, .3]$. The agent received a reward of 1 after reaching the goal. The state space was discretized into 231 states.

- The task in the cart-pole problem is to balance a pole on a moving cart. The power of the cart is random, uniformly distributed in the interval $[20, 75]$. The agent received a negative penalty when the pole deviated by more than 12° from the vertical. The state space of the cart was discretized into 5832 states.

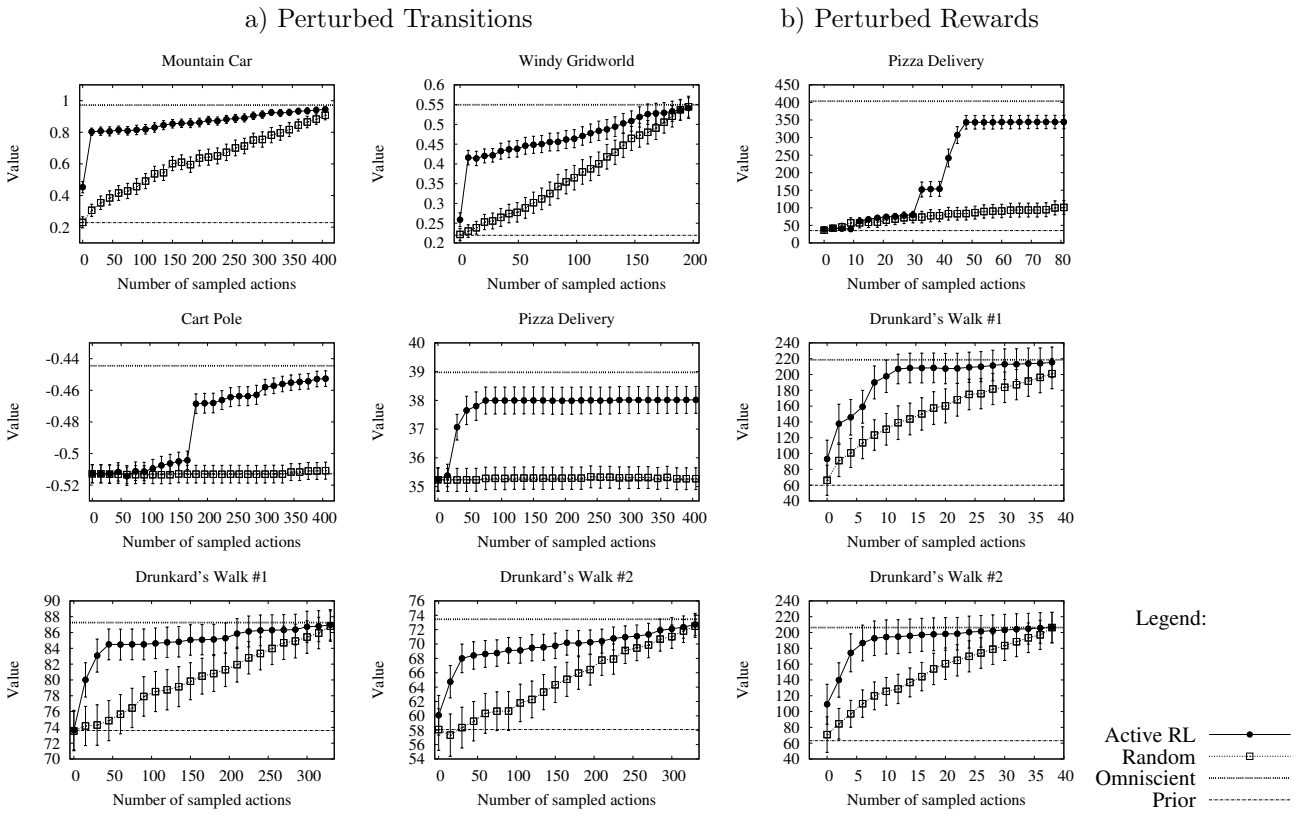


Figure 3.2.4: Evaluation of exploration strategies on perturbed MDPs. Error bars are based on 95% confidence intervals.

- Windy gridworld is a simple 10×7 gridworld with agent's movement affected by stochastic wind [70]. The agent's task is to find the optimal path to the goal from a specified initial state. This problem has 70 states.

- Pizza delivery problem is based on the racetrack example [70]. The agent drives a car by controlling its horizontal and vertical speed components. The agent received a reward of 100 for reaching its destination and a penalty of -5 for trying to drive off the road. In addition, the agent received random rewards and penalties uniformly distributed between 20 and -100 in randomly selected 2% of the states for either delivering a pizza or hitting a pothole. This problem has 4769 states.

- In the drunkard's walk problems, two 10×10 gridworld domains with random rewards and penalties were generated automatically. The agent can move in the four compass directions, but in each move it either moves in the intended direction or deviates diagonally from it (either to the right or to the left) with equal probability.

In the first set of experiments, the effectiveness of active reinforcement learning for transition probabilities was evaluated. The system was provided with an initial description of the problem, as given above. It then

performed the sensitivity analysis and sorted state/action pairs based on their sensitivity values. A different problem specification was then generated by randomly perturbing all the transition probabilities. This new specification represented the actual world in which transition probabilities are different from the expert-provided MDP specification. The agent was allowed to sample one action at a time in this actual world, replace user-specified transition probabilities with their maximum likelihood estimates (based on 10,000 samples), use an MDP solver to find the optimal policy in this "corrected" MDP, and evaluate this policy in the actual world[6]. We tested two different ways of selecting the order in which actions were tested: 1) the active RL agent which samples the actions in order of decreasing sensitivity, and 2) the random agent which samples actions randomly. For comparison, we also tested two agents applying fixed policies: 1) the prior agent which applies the optimal policy for the expert-provided MDP specification, and 2) the omniscient agent which knows the transition probabilities in the actual test world and selects the optimal policy for this world. The utility of the policy of each of these four agents appears in the plots in Figure 3.2.4-(a) as a function of the number of actions tested. The results are averaged over 100 different randomly generated actual worlds[7]. In each domain, the active RL agent outperforms the random sampling agent. Also notice that the prior agent which relies solely on the expert's specification performs poorly, indicating the need for exploration. The MDP structure for these problems is arbitrary and *does not* satisfy the conditions for rapid converges from Subsection 3.2.5. Thus, the experiment demonstrates that Newton's method converges reasonably well in practice even in cases for which we have not been able to derive strong theoretical guarantees.

In the next experiment, the random worlds were generated by perturbing the rewards rather than transition probabilities of the MDP[8]. Performance of the four exploration strategies on the three domains with nontrivial reward structure appear in Figure 3.2.4-(b). Once again, the active RL agent significantly outperforms the random sampling agent.

We also experimented with approximate active RL in the sailboat simulation, in which the agent must navigate a whirlpool of water current to reach the finish line. The whirlpool was modeled by ten concentric bands based on the distance from the center of the vertex, and the magnitude of the current varied proportionally to this distance. The expert-specified world reflected uncertainty about the direction of the whirlpool current: the direction of the current was counterclockwise with probability 0.1, clockwise with

[6]This exploration strategy assumes the ability to sample the world with resets.

[7]The test worlds were generated by perturbing transition probabilities of each action uniformly in the probability simplex within a radius of 0.6 around the expert-specified values T_0. A random variable $T \in \mathbb{R}^n$ uniformly distributed on the n-dimensional probability simplex can be generated from $n-1$ random variables $X_1, .., X_{n-1} \sim Uniform(0,1)$ by sorting them into $X_{(0)} \triangleq 0, X_{(1)}, .., X_{(n-1)}, X_{(n)} \triangleq 1$, and letting $T_i = X_{(i)} - X_{(i-1)}$ [26]. Rejection sampling is then used to ensure that $\|T - T_0\| \leq 0.6$.

[8]The test worlds were generated by perturbing all the nonzero rewards uniformly in the interval $[-70, 70]$ around the expert-specified values.

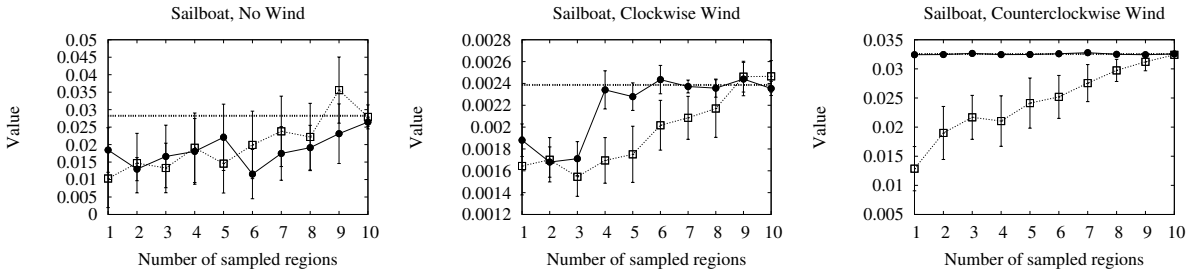


Figure 3.2.5: Evaluation of exploration strategies in the approximation architecture. The legend is the same as in figure 3.2.4, but with Prior strategy not shown (its value is too low to appear on the plots). Error bars are based on 95% confidence intervals.

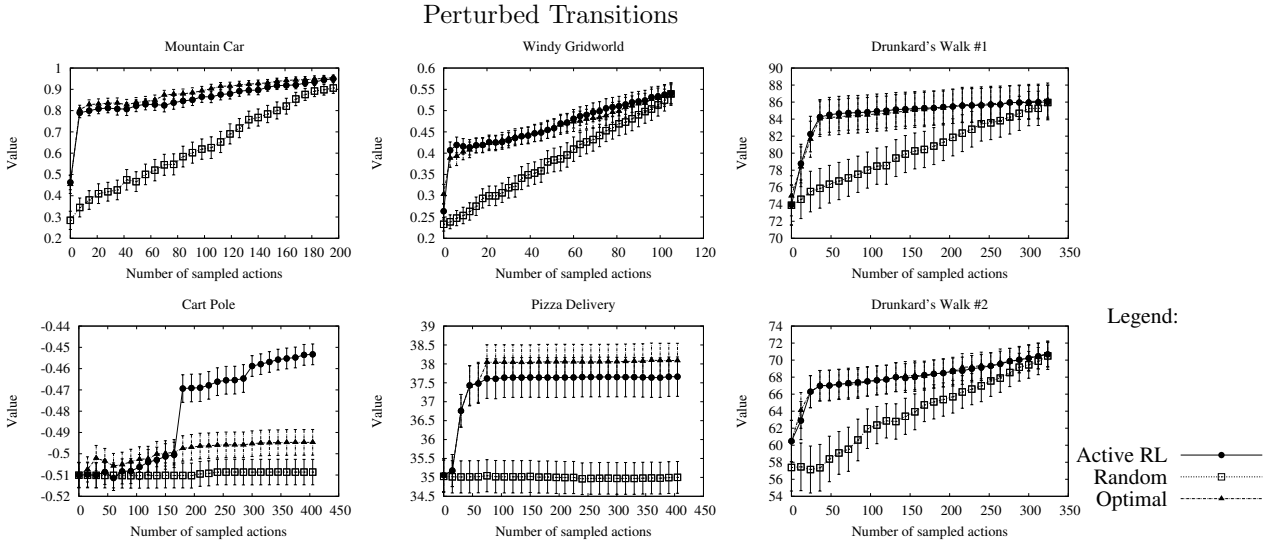


Figure 3.2.6: Comparing Active RL with Optimal Active RL (based on Hammersley's dense sampling). Error bars are based on 95% confidence intervals.

probability 0.1, and there was no current with probability 0.8. The agent received a positive reward of 1 upon reaching the finish line. In each iteration of the active RL algorithm, local policy search was performed from each of the seven policies shown in Figure 3.2.3, and the best policy was selected. After the sensitivity order for the ten bands was determined with approximate active RL, the sensitivity exploration strategy was tested in three actual worlds: one with a deterministic clockwise current, one with a deterministic counterclockwise current, and one with no current. The results appear in Figure 3.2.5. In the two worlds with current, the algorithm which samples bands according to the active RL-prescribed order outperforms the algorithm which samples bands according to a random order. In the world with no current, the performance of the two algorithms is similar. In these worlds, some policies discovered in estimated MDPs anomalously outperform the omniscient policy because the local policy search algorithm does not always discover the optimal policy.

Finally, we compared Active RL with Optimal Active RL (see Subsection 3.2.6). The results appear in Figure 3.2.6, which shows the performance of both algorithms (and random sampling, for comparison) as a function of the number of actions tested in the actual world. In these experiments, the prior in each of the problems was corrected by replacing prior transition probabilities for tested actions with correct transition probabilities in the actual world (instead of maximum likelihood estimates), and the optimal policy for this hybrid specification (prior for untested actions, correct for tested actions) was evaluated in the actual world. The performance of the optimal policy is plotted for the three methods for selecting the order in which the actions are tested. In most problems, the performance of Optimal Active RL is virtually identical to the performance of Active RL, demonstrating that our variant of Newton's method converges to a good approximation of the optimal solution in practice.

3.2.9 Conclusions

In this section, we presented a new algorithm for combining exploration with prior knowledge in reinforcement learning. We demonstrated that our algorithm can be implemented efficiently using policy iteration and a standard SOCP solver. We also introduced an approximate version of active RL to be applied in domains with large state spaces. In addition to being useful for exploration, the active RL algorithm can be used by an MDP designer to determine which regions of the state space require most precision in specifying transition probabilities and rewards. An important future extension of this work is designing a policy which explores the sensitive regions of the state space without resets.

3.3 Conclusions

In this chapter, we explored two ways of processing intuitive domain knowledge for reinforcement learning problems. The knowledge is expressed either in form of a complete specification of the world dynamics or in form of qualitative statements constraining the way in which the world behaves. In both cases, we presented efficient algorithms for processing the prior.

Chapter 4

Prior Knowledge and Compiler Optimization

4.1 Introduction

The application of high-level program transformations such as loop unrolling, array tiling, and software pipelining are critical in optimizing performance. Deciding how to apply these transformations can be challenging. These decisions must balance subtle interactions among characteristics of the underlying architecture, the source code, integration with other transformations, and so on. Optimizing compiler and code generators such as ATLAS [77] and FFTW [37], therefore, embody a decision procedure either explicitly or implicitly to resolve these choices. Intuitions (confirmed by decision theory) tell us that resolving such difficult choices satisfactorily requires a great deal of information.

Most commonly, this information is supplied explicitly via a prior performance model. Such models are extremely efficient, generating solutions almost instantaneously. But the information they embody comes entirely from their designer's formal idealization of the process to be optimized. It excludes phenomena that the designer believes to be negligible or too complex to analyze.

By contrast, an empirical approach collects information directly from the system on which the compiler or the code generator is deployed. This results in first-hand information which can be more accurate than that of a prior performance model. For example, many versions of a loop using different tilings paired with various loop unrolling amounts might be generated and executed. The combination with the best measured performance is then selected. Unfortunately, searching through combinations of parameter values can be hugely expensive. As a result, this approach cannot service the fast execution times usually expected from compilers as a prior performance model does. But empirical optimization is well suited to library generation where the high cost of optimal configuration decisions can be paid once. Well-known library generators that employ empirical optimization include FFTW [37], ATLAS [77], PhiPAC [10] and SPIRAL [80].

An alternative decision procedure is a *hybrid adaptive intelligent model(AIM)* which includes only the prior information from the designer which he or she is most confident of. The rest is then gleaned empirically. The prior partial model might answer some optimization questions directly but would also suggest which measurements are likely to be most informative and so guide and limit the empirical searches. The accuracy

of this decision procedure is rooted in first-hand measurement of the actual system to be optimized. But focused by prior information, the approach may eventually be efficient enough to make real-time optimization decisions and be automatically re-invoked when necessary to react to changing situations.

In this chapter, we describe the hybrid adaptive intelligent model we have developed. We evaluate our method by applying it to the optimization of the matrix matrix multiplication kernel generated by ATLAS and then comparing it with an analytic model and the empirical optimization approach used by ATLAS. Experimental fairness dictates that these three approaches be compared on equal footing. They must be applied to the same optimization task in as similar a setting as possible. To this end we use the matrix multiplication framework of ATLAS as our experimental platform but without its hand-tuned additions whose non-automated influences could be conflated with the behaviors we wish to monitor.

This chapter, however, is not so much about ATLAS nor about the optimization of matrix matrix multiplication, but about AIM. The main reason why we use ATLAS for the evaluation is that this is the only system for which both accurate models and efficient empirical search has been developed. Comparing with a system as higly tuned as ATLAS was a difficult challenge, but a necessary one to show the virtues of the new approach. We expect AIM to be applicable in a wide range of domains and particularly useful when the complexity of the algorithm precludes the development of accurate models.

As discussed in Section 4.5.2, for the case of ATLAS, the adaptive approach can perform better than the analytic model and is much more efficient than the empirical approach. The analytic model is based on an architectural idealization that cannot perfectly capture the actual machine to be optimized. On the other hand, the ATLAS routine samples broadly from a large but limited region of the parameter space that, on occasion does not contain the optimal configuration. The adaptive approach only samples those points deemed to be informative given the results of previous samples. This can greatly increase the range of parameter values it entertains, but it only does so when there is an expectation of optimization improvement.

In library installation efficiency is less crucial since cost can be amortized over the lifetime of the machine. But even here there are at least four situations in which efficiency can be important.

1) Adaptation may have to be applied at runtime, in which case an extensive search is not possible, and prior models (when available) may not be accurate enough. This type of search involves measuring the performance of various versions of pre-compiled code during the sampling phase of the executing, and then using the best version during the (much longer) production phase [27]. Note that runtime searching tailors the optimization system to the requirements of the user not available at library installation time (for instance, small blocking parameter values will be sele cted if the user only multiplies small matrices).

2) Efficient adaptation can be applied at the time of compilation. [75] describes a compile-time optimization

framework that employs empirical search which receives performance feedback from a fast estimator.

3) The space of possible versions can be too large even for once-in-a-life time installation. Empirical search complexity grows exponentially with the number of interacting optimization parameters.

4) An interesting application of library routines is as a benchmark to evaluate alternative machine designs. More efficient adaptation can enable a wider exploration of possible designs.

The work in this chapter has been published [32]. The chapter is organized as follows: Section 4.2 describes the search module of ATLAS. The model approach to optimization is discussed in Section 4.3. AIM, our adaptive hybrid approach, is presented in Section 4.4. Experimental results are shown in Section 4.5. Section 4.6 discusses other possible domains of application of AIM and presents related work. Finally, Section 4.7 presents our conclusions.

4.2 ATLAS

We focus on the optimization of the MMM routine, and use ATLAS for comparison purposes, since it is a well-known system that employs empirical search to optimize the MMM routine. ATLAS contains an generator search module and a multiple implementations search module. The generator search contains a code generator that outputs a kernel based on input transformations. This module searches the inputs that result in the best performing kernel. The multiple implementation module searches among hand-written codes for MMM kernels. ATLAS selects the best-performing kernel out of both modules. ATLAS also records results from previous installations on the target platform and can reduce the installation time by using these instead of the empirical search.

In this work, we focus on the generator search module. The search is used during the installation procedure to find the optimal values of code transformation parameters (amount of tiling, unrolling, etc.). It consists of: (1) generating the versions of matrix multiplication with the parameter values to be tested, (2) compiling and executing them, and (3) selecting the version that perform best.

In this Section we first examine the transformations that ATLAS applies to the MMM (Section 4.2.1). Then, we explain how ATLAS searches for the most appropriate parameter values of these transformations (Section 4.2.2).

4.2.1 Transformations

The code implementing MMM is shown in Figure 4.2.1. Several studies have found that computing this matrix multiplication using the library generated by ATLAS results in higher performance than that obtained

```
for (j = 1; j <= M; j++)
  for (i = 1; i <= N; i++)
    for (k = 1; k <= K; k++)
      C[i][j] = C[i][j] + A[i][k] * B[k][j]
```

Figure 4.2.1: Matrix Multiplication Code

when the naive MMM implementation in Figure 4.2.1 is compiled using a general purpose compiler. The reason for this performace gap is that compilers do not apply the appropriate transformations and/or they do not use the correct parameter values for these transformations [23, 81].

ATLAS restructures the code in Figure 4.2.1 by applying blocking and pipeline scheduling.

- Blocking: This transformation can be accomplished by applying a tiling loop transformation [78]. ATLAS applies blocking at the cache and the register levels:

- Cache Blocking: ATLAS uses tiling to decompose the matrix multiplication of large matrices into the multiplication of smaller sub-blocks. The size of each sub-block is $NB \times NB$, where NB is an optimization parameter that needs to be chosen so that the working set of the tiled MMM fits in the cache [14, 21, 78]. We call the resulting code mini-MMM.
- Register blocking: The mini-MMM code itself is tiled to optimize the utilization of the registers. The resulting code, that we call micro-MMM, multiplies a column of MU elements of matrix A by a row of NU elements of matrix B and stores the result into a $MU \times NU$ sub-matrix of C. MU and NU are optimization parameters that must be chosen so that $MU + NU + MU \times NU$ fit in the registers of the processor [6].

To improve register allocation, ATLAS uses scalar replacement [16]: and copies the $NB \times NB$ sub-matrices to consecutive memory locations. This strategy reduces the number of cache and TLB misses. Additional transformations such as loop unrolling and load scheduling applied in ATLAS are described in detail in [77, 81, 82].

Table 4.1 shows a summary of ATLAS transformations.

Transformation	*Parameter Name*
L1 cache blocking	$NB \times NB$
Register blocking	MU and NU
Inner loop unrolling	KU
Multiply-add scheduling	*Latency*
Load Scheduling	*Ifetch,Nfetch,Ffetch*

Table 4.1: Summary of the optimization parameters.

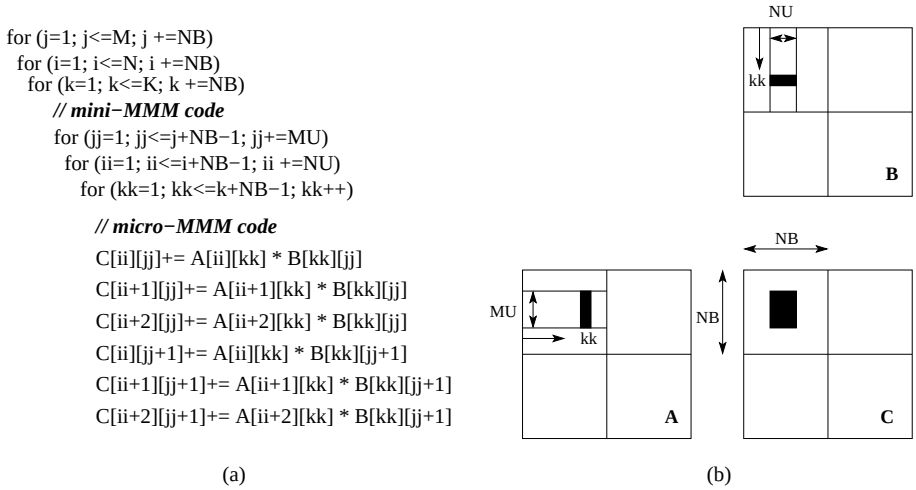


Figure 4.2.2: MMM after cache blocking and register blocking. (a) Code where the innermost loops are unrolled by a factor of *MU=2* and *NU=3*. (b) Picture of the code on the left.

4.2.2 Search

ATLAS performs an extensive search of the parameter values presented in the previous Section. Since ATLAS searches for several parameters, when searching for one parameter, ATLAS needs to assign values to the other parameters it has not yet optimized. These values are initially assigned based on results obtained from the execution of benchmarks which estimate hardware parameters such as cache size and number of registers. After a parameter is optimized, the value that obtains the best performance is used for the search of the subsequent parameters. Parameter values are searched in the same order that appears in Table 4.1. For the L1 cache blocking, ATLAS generates versions of the mini-MMM code with a matrix size $NB \times NB$, where NB varies from 16 to the minimum of (80 and $\sqrt{L1\ Size}$), in steps of 4. For the rest of the parameters, the search range can be found in [77, 81], where a more detailed explanation of the search process is also presented.

4.3 Model

In [81] the notion that empirical optimization is more effective than model-driven optimization is challenged by demonstrating that a model-based optimization strategy can calculate near-optimal parameter values without incurring the sampling cost of empirical search. We use Yotov's model as our initial guess of the parameter values. We also compare the experimental results obtained by our approach with the results obtained by Yotov's model. A detailed description of the model can be found in [81].

The model depends on accurate estimates of machine parameters that include the L1 cache and line

size, the number of registers, the latency of the multiply instruction, the existence of a fused multiply-add instruction, and the number of functional units.

1. L1 cache blocking ($NB \times NB$): The idea of the model is to compute the value of NB that optimizes the use of the L1 data cache. The model is based on the memory access trace of the mini-MMM, and takes into account the loop order, L1 cache and line size, and the LRU replacement strategy of caches. This analysis finds that for a JIK order, the optimal value for NB is the maximum value of NB that satisfies the inequality below:

$$\left\lceil \frac{NB^2}{L1\,Line\,Size} \right\rceil + 3 * \left\lceil \frac{NB}{L1\,Line\,Size} \right\rceil + 1 \leq \frac{L1\,Size}{L1\,Line\,Size}$$

2. Register blocking (MU and NU): The model uses the formula obtained for the L1 cache blocking, taking into account the specific features of the register file, such that it has unit line size. The model selects the maximum values of MU and NU such that $NU \approx MU$ and $MU \times NU + MU + NU + Latency \leq Number\ Of\ Registers$

3. Inner loop unrolling (KU): The approach is to unroll the loop as much as possible, within the constraint of not overflowing the L1 instruction cache. In most cases, fully unrolling the loop ($KU=NB$) works well.

4. Multiply-add Scheduling($Latency$): The value of $Latency$ is based on the number of functional units and their latencies. The value of these hardware parameters is detected by a benchmark.

5. Load Scheduling($Ifetch, Nfetch, Ffetch$): The model sets these parameters to $Ifetch=Nfetch=2$, and $Ffetch=1$

The model we just presented mimics ATLAS, so it computes a blocking value for the L1 cache. However, sensitivity analysis reported in [81] and other experiments have shown that in some machines, blocking values that overflow the L1 cache obtain better performance. The conjecture is that in these machines, the large block value that results in the best performance corresponds to the blocking value for the L2 cache. Blocking for L2 may result in higher performance than L1 because in out-of-order processors, which have a deep pipeline, the latency of accessing the L2 cache can usually be hidden without stalling the processor. The rationale is that the processor can proceed executing instructions that do not depend on the missed data. A larger blocking value also increases the opportunity for higher ILP and for the compiler to reorder instructions [19]. However, notice that tiling for L2 may not always be the best choice, because large tiles can result in more time spent in the cleanup code, which can degrade performance for some of the codes calling the MMM library generated by ATLAS [1]. However, it has been shown that in some cases it is necessary to tile for L2 [1], and this is confirmed by our experiments (Figure 4.5.1) where the MMM library generated by ATLAS is evaluated in the contex of matrix-matrix multiplication.

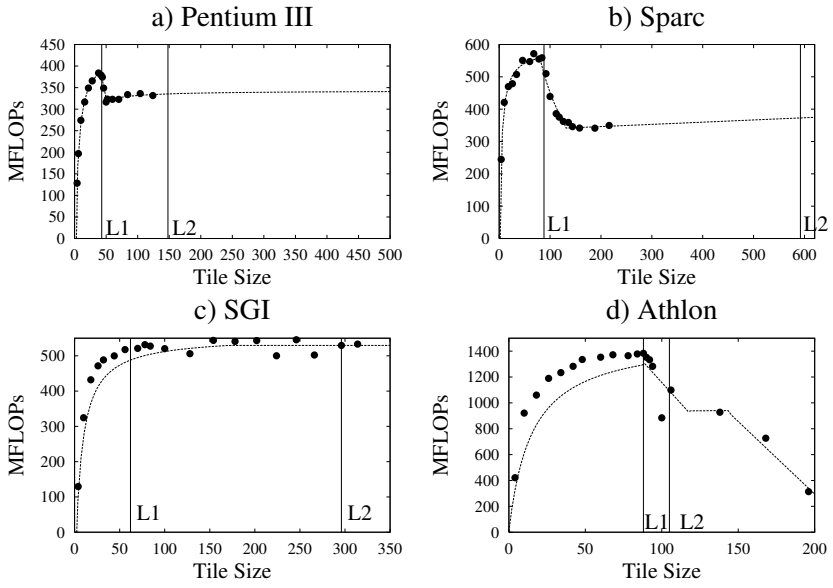


Figure 4.4.1: Performance as a function of cache block size NB (complete instruction cache unroll: KU=NB)

Given that some architectural platforms perform better when blocking for the L2 cache, we extended the model from [81] to estimate an appropriate L2 blocking parameter value. The inequality above used to compute the L1 cache blocking factor cannot be used to compute the L2 cache blocking factor because it does not take conflict misses into account. Thus, to compute the L2 blocking factor we use a conservative approach that ensures that NBxNB blocks of data from all three matrices A, B, and C fit in the L2 cache. This happens when the combined size of these three blocks ($3 * NB^2$) is equal to the size of the L2 cache.

For ATLAS (that performs an exhaustive search for the L1 blocking factor), it would be very costly to extend the search beyond $\sqrt{L1\,Size}$, to consider L2 blocking factors. A model can compute the blocking factors for the two caches, but it is based on strong assumptions that may not hold equally well on all the architectures. Our adaptive approach can correct the model-provided estimates of the blocking factors and decide which cache (L1 or L2) to tile for.

4.4 Adaptive Modeling

Our adaptive approach combines the information embedded in the model from Section 4.3 with feedback information obtained from the execution of versions of the mini-MMM code. Both types of information are used to search for the maximum of the mini-MMM performance function. In our approach, the shape of this function is determined through experimentation. Each experiment consists of generating, compiling, and executing the mini-MMM code. The mini-MMM code is generated by ATLAS's code generation module,

ensuring that the space of available transformations is the same for ATLAS search and AIM. The feedback provided by each experiment (in form of mini-MMM performance) is used to design subsequent experiments to maximize information about the location of performance-maximizing parameter values. Maximizing performance can be done either via a local search (e.g., by performing hill climbing) or by modeling the whole performance function globally via appropriately chosen *regression curves*. In our algorithm, we use both global and local search techniques.

We rely on the assumption that cache and register blocking parameters can be optimized independently of each other (orthogonal search). Both ATLAS search and the model make this assumption as well. Our algorithm is as follows:

1. Optimize the cache blocking parameters *(NB, KU)*: We first optimize the cache blocking parameters and set the register blocking parameters (MU,NU and Latency) to the values chosen by the model. We use global search and determine the location of the maximum from the shape of the regression function by analyzing the general shape of the plot of mini-MMM performance. Figure 4.4.1, for example, shows sampled data collected on four different machines. In each plot, the points show the performance of the mini-MMM code (Y-axis) for different values of cache block size (X-axis). As these sampled points are being collected, a regression curve is fitted to the data (this curve is shown in Figure 4.4.1 as well). The shape of the curve is adjusted with each newly collected sample point. The best values of the optimization parameters can be determined directly from the location of the maximum point on the regression curve. In this case the prior knowledge from the model is used to indicate to the family of regression curves that the maximum performance is going to be located in the neighborhood of the model-predicted values. Our optimization algorithm is described in detail in Section 4.1.

2. Optimize the register blocking parameters *(MU, NU, Latency)*: Register blocking parameters are optimized using the cache blocking parameters found to be optimal in step 1. The algorithm to optimize the register blocking parameters is very simple: it performs hill climbing in the space of register blocking parameters starting at the parameter values calculated by the model.

Optimization in step 1 requires a sophisticated algorithm because multiple levels of the cache hierarchy introduce multiple local maxima in the performance function. For example Figure 4.4.2 shows the performance obtained by the mini-MMM code as the tile size increases. On Pentium III, the figure shows two distinct peaks, each corresponding to blocking factors for L1 and L2 caches. Optimizing register blocking parameters is a much simpler problem than cache blocking, since there are no multiple levels of cache to worry about, and the set of reasonable parameter values is small with a well-defined global optimum. For

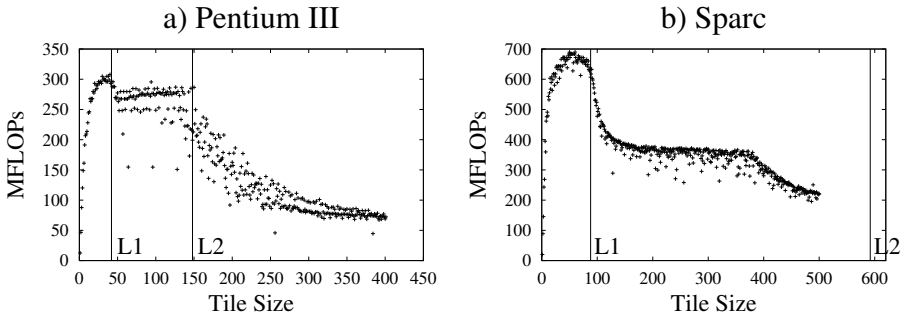


Figure 4.4.2: Complete sampled performance curves on two machines. The vertical lines correspond to the blocking factors for L1 and L2 as predicted by the model

this reason, a simple hill climbing algorithm suffices for step 2. We focus our discussion in this chapter on the harder problem of optimizing the cache blocking parameters.

4.4.1 Cache blocking parameters

AIM constructs a nonlinear regression curve representing the sampled performance of the minim-MMM code as a function of tile size, with register blocking parameters being held constant. Figure 4.4.3 shows examples of such curves fitted using some data collected on the Pentium III in 4.4.1-(a). The regression curve has different segments, which means that different regression functions are used to fit the data in the distinct regions of performance. The resulting final regression curve that fits the whole data sample is obtained from the regression functions computed in each region.

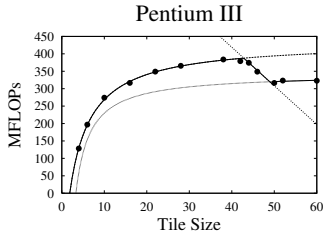


Figure 4.4.3: Piecewise regression: Zoomed-in section of Figure 4.4.1-(a)

Our algorithm has two main strategies:

1) The first strategy uses the regression curve available at a given time to identify the next experimental point. That is, it identifies the size of the tile that will be used for the next sample $(tilesize_n, performance_n)$. The goal is to sample the point that provides the best feedback about the location of the maxima. This strategy is called Active Sampling, and is explained in Section 4.4.1.

2) The second computes the curve that "best" fits a set of experimental points, taking into account the

prior information provided by the model from Section 4.3. This strategy is done using the Maximum a Posteriori Bayessian Estimate, and is explained in Section 4.4.1

Our algorithm consists of a loop, that in each iteration applies strategy (1) and generates a sample point and then applies strategy (2) to compute the best fit given all the points selected so far. As the search is conducted, each sample point is determined by generating a mini-MMM program based on the tile size determined by the strategy (1), compiling the program, and measuring the program's execution time.

Maximum a Posteriori Bayessian Estimate

Given a set of experimental points, the second strategy computes a curve that is a good fit to these points and, at the same time, to the model (known as prior). In our case, good fit to the model means that the two peaks of the resulting curve are not too distant from the values predicted by the model. We now describe this strategy more formally.

Let β be one of the curves identified by our algorithm. This curve consists of four segments of the form $(\frac{a\times x+b}{c\times x+1})$ [1], and therefore is defined by a set of 12 coefficients (called w), and the separators $(l1, lm, l2)$ that correspond to our two peaks ($l1$ and $l2$) and the valley between the two peaks (lm). Initially $l1$ and $l2$ are at the values L1 and L2 predicted by the model, and lm is a value between them. In successive iterations of our algorithm, the values of $l1$, lm, and $l2$ are determined in the process of maximizing the formula given below.

Given a set of experimental points D, the maximum a posteriori Bayesian estimate is used to determine the best curve $\widehat{\beta}$ that maximizes the probability density function $P(\beta|D)$. This density function can be expressed using Bayes rule:

$$P(\beta|D) = P(D|\beta) \times P(\beta)/P(D) \tag{4.4.1}$$

$P(\beta)$ is known as the prior and in our case incorporates information from the model. Since any curve β is identified by the w coefficients and the triple $(l1, lm, l2)$, we can say that $P(\beta) = P(w, (l1, lm, l2))$. Then, $P(w, (l1, lm, l2)) = P(w|(l1, lm, l2)) \times P(l1, lm, l2)$. Now, we assume that the curves of the form we have selected with peaks at $l1$ and $l2$ and the valley at lm have a uniform distribution and therefore the density function $P(w|(l1, lm, l2))$ is a constant. We also assume that the three-dimensional random

[1] It is possible to extend the curve with more segments to model architectures with three or more levels of memory hierarchy, although this was not necessary for the machines we examined.

variables $(l1, lm, l2)$ have a normal distribution $N(\begin{bmatrix} L_1 \\ L_m \\ L_2 \end{bmatrix}, \begin{bmatrix} \sigma_1^2 & 0 & 0 \\ 0 & \sigma_m^2 & 0 \\ 0 & 0 & \sigma_2^2 \end{bmatrix})$, where σ_1^2, σ_m^2, and σ_2^2 are user-controlled parameters representing one's confidence in the model's prediction.

The other term of the equation, $P(D|\beta)$, is computed assuming that the errors have a normal distribution and therefore is computed using the equation in 4.4.2. This term computes the total squared error of n data points $(tile\ size_1, performance_1)...(tile\ size_n, performance_n)$ of the sample D versus the n data points $(tile\ size_1, \beta(tile\ size_1))...(tile\ size_n, \beta(tile\ size_n))$ of the regression curve β, assuming independent identically distributed gaussian noise. This term favors the curves β that fit the sample well.

$$P(D|\beta) = (\frac{1}{\sqrt{2\pi\sigma^2}})^n e^{-\sum_{i=1}^{n} (performance_i - \beta(tile\ size_i))^2/(2\sigma^2)} \tag{4.4.2}$$

Notice that $P(l1, lm, l2)$ favors the curves that agree with the model. As the sample size increases more points contribute to the total squared error and penalize the curves that do not fit the data more heavily, while $P(l1, l2, lm)$ remains unchanged. Thus, from the form of equation 4.4.2 it is clear that the system converges to the best regression curve in the limit even if the prior information is inaccurate, but this convergence happens much faster when the model is good.

Active Sampling

The main source of efficiency of AIM comes from its ability to select informative sample points intelligently. For that, it must take into account conflicting objectives: reducing the time to collect the sample and selecting the most informative points. The first objective directs the system to sample points close to the origin, because the sampling time increases with increasing tile size NB (and the amount of unrolling KU) due mainly to the significant increase in the amount of time required to compile the program.[2] The second objective is to select the points that provide more information about the location of the peak of the function.

To reconcile these objectives, a heuristic that simulates potential fields is used. It places a negative charge at each sample point to discourage oversampling in the same region and a positive charge at the origin to encourage less time-consuming data points (since programs generated with smaller values of cache blocking/unrolling take less time to compile). Positive charges encourage sampling in the region around

[2]With bigger tile sizes, the size of the completely unrolled register loop nest increases, forcing the optimizing compiler to spend more time on instruction scheduling. Increasing the cache block size from 40 to 400 on the SGI machine increases compilation time from 4 seconds to 4 minutes.

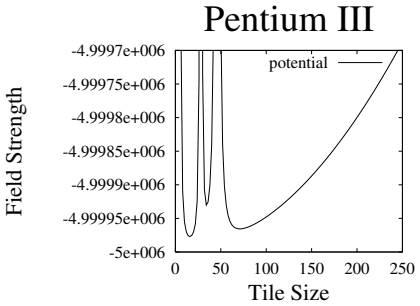


Figure 4.4.4: Potential Field for Active Sampling. The field is constructed based on three sample points. It increases away from the origin and at previously sampled locations.

them, negative charges have the opposite effect. A positive charge is also imposed on regions contributing information about the highest peak. This charge is proportional to the estimated probability that the peak that appears to be the highest actually is the highest. The point that minimizes the potential field is selected for sampling.

An example of the heuristic can be seen in Figure 4.4.4. The potential field $U(x)$ is a function of the tile size x. The system computes $U(x)$ for every tile size x and chooses the tile size with minimum potential energy for sampling. Tile sizes with low potentials experience the least amount of repulsive force and the greatest amount of attractive force. The potential field is calculated as a sum of contributing factors. Each previously sampled tile size y contributes $\frac{\nu}{(x-y)^2}$ to the potential field at x, creating a repulsive force that increases at tile sizes x close to the sampled point y. The attractive field at the origin contributes $\xi * (x-0)^2$ to the potential field at x, resulting in an attractive force that decreases with increasing tile size. ν and ξ are user-defined constants controlling the strengths of the forces creating the field. The advantage of using this heuristic is its efficiency in combining multiple objectives.

Deciding which peak dominates

AIM also needs to determine which peak dominates to decide where to collect the next sample. For the sampled data, many regression curves are possible. Each of these curves has some probability of being the correct one. Thus, there is a probability distribution over regression curves. The height of each peak is a function of the parameters β (Section 4.1.2) of the regression curve. Hence, for each peak, there is also a probability distribution over its height. Let $h_1(\beta)$ be the height of the first peak and $h_2(\beta)$ be the height of the second peak. These functions have a joint probability distribution ($P(h_1(\beta) = x, h_2(\beta) = y)$) that is the probability that the height of the first peak in the correct regression curve is x, while the height of the second peak is y. The probability that the first peak in the correct regression curve is higher than the second peak is given by the mass of the joint probability distribution in the region $x > y$. Thus if we knew the shape of this

joint distribution function, we could compute the domination probability ($P(h_1(\beta) > h_2(\beta))$). The shape can be estimated with standard statistical techniques. In particular, if $\widehat{\beta}$ represents the parameters of the best maximum a posteriori curve picked based on some sample of data points D, as described in Section 4.4.1, the distribution of any function of the best curve $g(\widehat{\beta})$ is approximately Normal with variance proportional to $\frac{\partial g(\widehat{\beta})}{\partial \beta} * (H(\widehat{\beta}))^{-1} * (\frac{\partial g(\widehat{\beta})}{\partial \beta})'$, where $H(\widehat{\beta})$ is the Hessian of the regression function (for proof of this result, see [66]). We can take $g(\widehat{\beta})$ to be any function of the regression curve whose distribution we want to know. In particular, choosing h_1 and h_2 to be $g(\widehat{\beta})$ gives us the distribution we need above. It is approximately Normal, centered at the peak values computed from the parameters of the best regression curve $\widehat{\beta}$. As more data is collected, the variance of the Normal distribution decreases and the system becomes more confident that the heights of the peaks computed from the best regression curve are correct. This concentrates the mass of the joint distribution around the values computed from the best-fit regression curve. So, if we take the Pentium III machine (Figure 4.4.1-(a)), since the height of the L1 peak computed from the best-fit regression curve is higher than the height of the L2 peak, this concentration of mass, in turn, increases the probability that the L1 peak is the dominant one. Figure 4.4.5-(a) shows how this probability increases as the sample size grows.

The potential field is adjusted to encourage sampling in the region around the dominant peak, e.g., the first peak on the Pentium III. The amount of this adjustment depends on the computed domination probability. $P(h_1(\beta) > h_2(\beta))$ is used to measure the system's confidence in the assertion that the first peak dominates. The more confident the system gets, the more it discourages sampling in the dominated region (L2 region on Pentium III).

The above computation can also be used to estimate the system's uncertainty in the location of the dominant peak. Let $l_1(\widehat{\beta})$ be the function that computes the location of the first peak of the best-fit regression curve $\widehat{\beta}$. Taking $g(\widehat{\beta})$ above to be $l_1(\widehat{\beta})$, we find the distribution of the location function. The variance of this distribution can subsequently be used as a measure of uncertainty about the location of the first peak (Figure 4.4.5-(b) shows the uncertainty as a function of the sample size). This uncertainty decreases as more data is collected and the mass of the distribution concentrates around the location of the L1 peak calculated from the best-fit regression curve.

Finally, notice that the SGI machine has a different performance profile than the Pentium III (see Figure 4.4.1-(c) and (a)), what results in a different sampling behavior. In SGI, the optimal cache block size must take advantage of the L2 cache. After the system determines that the dominant peak lies beyond the L1 saturation point, it attempts to collect as much information as possible to ascertain how to take advantage of the L2 cache, even at the expense of incurring a higher sampling cost. It does not make any

a)Probability that the L1 peak is higher than the L2 peak

b)L1 peak location uncertainty

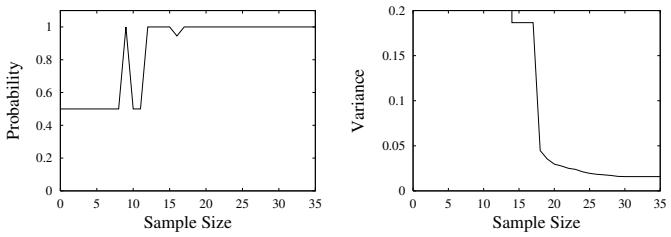


Figure 4.4.5: Pentium III installation time statistics.

sense to sample at lower tile sizes if these points do not provide any information about the predicted optimal peak of the model.

4.5 Experimental Results

In this section, we evaluate AIM, our adaptive optimization algorithm. The environmental setup used for our experiments is discussed in Section 4.5.1 and performance results are shown in Section 4.5.2. Our experiments are designed to evaluate the following claims:

1) AIM is more accurate than the analytic model.

2) AIM is faster than ATLAS search.

3) AIM scales better than ATLAS search as the complexity of the memory hierarchy increases.

4.5.1 Environmental Setup

Four different architectural platforms were used to determine the efficacy of our approach: Ultra Sparc III, Intel PIII-Xeon, SGI R12000, and AMD Athlon MP. Table 4.2 lists the salient architectural parameters of each platform.

The following algorithms were executed on each platform:

1) Model: We use the model from [81] described in Section 4.3. The model assumes that tiling for the L1 cache is optimal.

2) ATLAS search: This is the search strategy using the code generator as described in Section 2. ATLAS assumes that tiling for the L1 cache is optimal.

3) AIM: This is the approach presented in this chapter, as described in Section 4.4. For the user-controlled parameters σ_1, σ_m and σ_2 , we used the values 10, 25, and 25, respectively, while for the initial value of

	Sparc	Pentium III	SGI	Athlon
CPU	Ultra Sparc III	PIII-Xeon	R12000	AMD Athlon MP
Frequency	750 MHz	550 MHz	300 MHz	1533 MHz
L1d/L1i Cache	64 KB/32 KB	16 KB/16 KB	32 KB/32 KB	64 KB/64 KB
L2 Cache	8 MB	512 KB	2 MB	256 KB
Memory	4 GB	1 GB	512 MB	767 MB
OS	SunOS 5.8	Red Hat 7.3	IRIX64 v6.5	Linux 2.4.20-28.9smp
ATLAS Compiler	Workshop cc v5.0	gcc v3.2	MIPSPro cc v7.30	gcc v3.2.2
ATLAS Compiler Options (1)	-dalign -fsingle -xO2 -native	-fomit-frame-pointer -O	-O3 -64 -OPT:Olimit=15000 -TARG:platform=IP30 -LNO:blocking=OFF -LOPT:alias=typed	-fomit-frame-pointer -O
Fortran Compiler	Workshop Fortran 77 v5.0	g77 v3.2	MIPSPro Fortran 77 v7.30	g77 v3.2.2
Fortran Options	-daling -native -xO5 -pad	-O3 -fno-inline -funroll-all-loops -funroll-loops	-O3 -64 -OPT:Olimit=15000 -TARG:platform=IP30	-O3 -fno-inline -funroll-all-loops -funroll-loops

Table 4.2: Test Platforms. (1) ATLAS compiler and options are the defaults that ATLAS selects in each target platform.

l_m we used the tile size that the model predicts for L1 + 11 (Section 4.4.1). These values are determined empirically for one machine and then used without changes for all the others. Unlike ATLAS, which uses an average of three samples to estimate performance for each tile size, AIM only samples each tile size once and takes care of noise in the estimates via regression. The search for the optimal cache blocking parameter values terminates after collecting 20 points. Details about the stopping criterion are discussed below.

4) Extended ATLAS: This is an extension of ATLAS search that still performs a near-exhaustive search of the cache tile space, but considers the tile sizes that fit into either the L1 cache or the L2 cache. Extended ATLAS searches the cache tile space from 16 to the minimum of (80 and $\sqrt{L1\,Size}$), in steps of 4. Then, it searches from the minimum of (80 and $\sqrt{L1\,Size}$) to 500, in steps of 8. This extension of ATLAS search was implemented by us to test ATLAS's scaling properties and to compare its performance results with those of AIM, since AIM searches in the L2 area when it estimates that the L2 peaks dominates L1 peak (Section 4.4.1). The search for the register-tiling parameter values in Extended ATLAS is the same as in ATLAS search.

In addition, where appropriate, we provide for completeness the MMM performance for:

1) Full ATLAS: Complete ATLAS distribution which may use hand-written codes and parameter values from previous installs.

2) Compiler: Performance of native Fortran compiler shown in Table 4.2. The input to the Fortran compiler is the triply-nested loop shown in Figure 4.2.1

3) BLAS: Vendor supplied BLAS libraries.

All of the search strategies are integrated with ATLAS version 3.4.1. Each search strategy optimizes

	Model					ATLAS search				
	Lat	NB	MU	NU	KU	Lat	NB	MU	NU	KU
Sparc	5	88	3	2	88	6	68	2	2	68
Pentium	3	42	2	1	42	5	40	2	1	40
SGI	6	62	4	4	62	3	64	4	4	64
Athlon	4	88	1	1	88	1	72	3	1	72

	AIM					Extended Atlas Search				
	Lat	NB	MU	NU	KU	Lat	NB	MU	NU	KU
Sparc	4	60	2	2	60	6	68	2	2	68
Pentium	2	42	4	1	42	5	40	2	1	40
SGI	1	170	4	4	170	1	348	3	5	1
Athlon	1	88	4	1	88	1	88	4	1	88

Table 4.3: Discovered Optimal Parameters

	Model	AIM	ATLAS	Extended
Sparc	376.66	851.04	832.63	835.56
Pentium	384.70	379.04	409.43	414.15
SGI	499.81	553.15	505.4	561.82
Athlon	764.88	1650.08	1529.54	1650.08

Table 4.4: Mini-MMM Performance Comparison (in MFLOPs)

performance by generating versions of code (mini-MMM) with the parameter values under test, compiling and executing them. Once the optimal tiling, unrolling and scheduling values are found, a library is generated that uses the discovered values to multiply user-provided matrices. While it is plausible that optimal mini-MMM performance will translate into good performance when multiplying arbitrary matrices, this is not guaranteed. In this section we generate libraries for multiplying double-precision floating point numbers. For each algorithm and each platform under test, the following measurements are made:

- The amount of time needed to find the optimal parameter values.
- Performance of mini-MMM code generated with the values found to be the optimal.
- Performance of the generated library on a wide range of matrix sizes.

4.5.2 Experimental Results

Table 4.3 lists the optimal parameter values discovered by each algorithm. [3] The performance of the generated mini-MMM code based on these values is reported in Table 4.4. ATLAS search is always outperformed by the extended ATLAS search, since the latter conducts a more extensive search of a larger set of *NB* values that correspond to blocking for the L1 and the L2 caches. ATLAS search, on the other hand, limits its search range to block sizes that fit into the L1 cache. The model described in Section 4.3 is always outperformed

[3] On the Sparc, 32 registers are available, but the `-native` compiler flag that we are using directs the compiler to only use 16. This results in lower optimal values for MU,NU, and Latency.

by the original ATLAS and its extended version. Notice that the model computes the optimal block size exclusively for the L1 cache.

AIM performs well on all four platforms. On three out of four architectures AIM outperforms the model. Only on Pentium III AIM performs slightly worse (384 vs. 379 Mflops) than the model. By analyzing the parameters discovered by each strategy (Table 4.3) we can see why our adaptive approach performs better than the model. For the SGI, the highest performance is obtained when blocking for L2 (only Extended ATLAS and AIM find this result). For the Sparc machine the differences in tile size (88 in the Model versus 60 in AIM) account for a performance difference of about 10%. On Sparc, the difference in performance between the Model and AIM is mostly due to the optimal setting of the register blocking parameters (MU, NU, and $Latency$). The Model computes the register blocking parameter values assuming 16 registers, which are the registers used by the compiler. Unfortunately the compiler is not efficiently allocating the 16 registers, and as a result the register blocking values computed by the model result in register spills and lower performance. All the other approaches find smaller values for MU, NU and $Latency$ (Table 4.3) by using feedback information from the empirical search. Notice that on the Sparc 32 registers are available to the compiler. However we noticed that the `-native`[4] flag that we used directs the compiler to only use 16 registers. To fix this problem, instead of the ATLAS compiler options we could have used the `-xarch=v9a` flag that corresponds to the architecture of our target Sparc machine. However, we used the `-native` flag since it is the one that ATLAS selects by default.

For Athlon, the model finds the correct tile size but fails to find the right register blocking parameters[5]. For Pentium III AIM performs worse than the other approaches because the hill-climbing that we used to search for the register blocking parameters got stuck in a local optimum.

Overall, our adaptive approach performs, on the average, 62% better than the model, and on two platforms (Sparc and Athlon) outperforms the model by a factor of two. Our results also demonstrate that, by combining the model-based approach with our intelligent search strategy, we can achieve the same level of performance as the near-exhaustive search implemented in ATLAS and extended ATLAS search.

Figure 4.5.1 shows the performance of the libraries generated using the parameters in Table 4.3 for each of the optimization algorithms under study. Figure 4.5.1 shows the performance of each library as the size of the matrices being multiplied increases from 100 $\times$ 100 to 3000 $\times$ 3000. The Figure shows that there is a strong correlation between mini-MMM performance and performance of the final generated library, the metric that the end user of the system is interested in. For completeness, the Figure also shows performance

[4]The `-native` flag should direct the compiler to optimize the code for the current machine, but apparently the code generated when using this flag corresponds to that of an older architecture.

[5]Model can do a better estimation of the register blocking parameters by using a refined model for out-of-order execution and small number of logical registers (See [82])

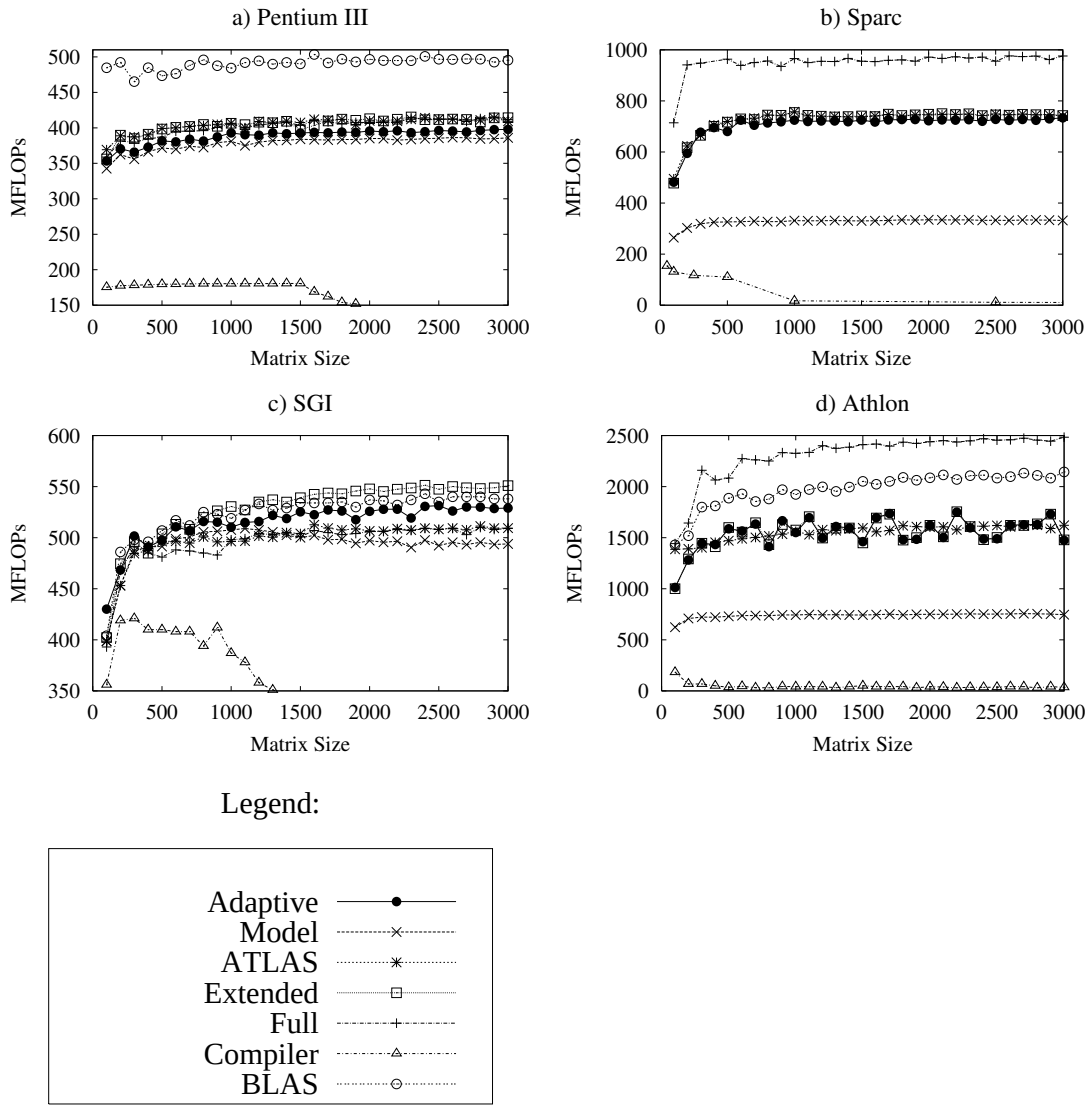


Figure 4.5.1: Library Performance Comparison for ATLAS Search, Extended Search, Model, AIM, Full ATLAS, BLAS and the fortran compiler. BLAS results are not shown for UltraSparc III. We contacted the vendor and we were informed that BLAS is not optimized for the specific architecture of our UltraSparc III. Results for compiler are only shown when its performance is higher than the MFLOPs shown in the Y-axis of each plot.

results for the library produced by Full ATLAS and the vendor supplied BLAS libraries, and the Fortran compiler. BLAS library is hand-tuned and, therefore, produces excellent results. The fortran compiler performs poorly, underlining the need for platform-targeted optimization.

Table 4.5 presents the amount of time required for each search strategy to complete. The model performs simple calculations and, therefore, takes a negligible amount of time to complete. AIM takes significantly less time than ATLAS search, justifying our second claim. The data in this table also demonstrates the poor

	Model	AIM	ATLAS	Extended
Sparc	0:00	3:12	8:59	36:32
Pentium III	0:00	1:49	5:30	43:50
SGI	0:00	14:02	59:00	237:01
Athlon	0:00	3:01	17:18	94:50

Table 4.5: Time to Complete the Search (in minutes)

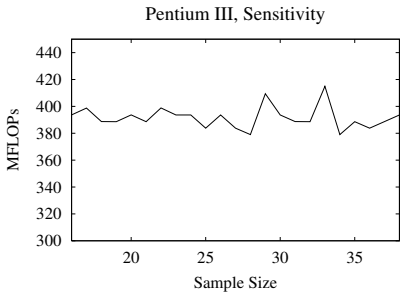


Figure 4.5.2: Mini-MMM Performance as a Function of Stopping Criterion on Pentium III.

scaling properties of ATLAS. Notice that extending the range of cache block sizes considered by ATLAS search to take tiling for the L2 cache into account increases the search time at least by a factor of four on most machines and a factor of eight on Pentium III. The adaptive approach, on the other hand, takes less time to complete than the original ATLAS search (that searches for L1 tiling parameters only) even though its search range is equivalent to that of the extended ATLAS search (tiling either for L1 or L2). In particular, notice that the search time of AIM versus ATLAS (14:02 versus 59 minutes, respectively) on the SGI machine, where AIM also needs to run experiments in the more time consuming points in the area of the L2 peak. This observation supports our third claim.

Finally, we evaluate our stopping criterion. Empirically, we have determined that a sample size of twenty points produces sufficiently accurate estimates of the optimal parameter values. How robust is our algorithm with respect to this assumption? A typical sensitivity plot is presented in Figure 4.5.2. It shows the optimal mini-MMM performance as a function of the stopping criterion (varying the number of points in the collected sample) for the Pentium III architecture. The plot shows no significant improvement in performance with increasing sample size. However, it also shows that a stopping criterion of sixteen points would suffice for this architecture.

4.6 Conclusions

Recent work closest to ours include a study of hill climbing to select the best parameter values for certain transformations of the matrix matrix multiplication [83] and a hybrid approach that uses a compiler model

to focus the range of the search on a small number of candidate implementations, which are subsequently tested empirically [20]. This approach was also tested on matrix multiplication, and the main difference with ours is that in [20], a model is applied only once prior to the empirical search to prune the search space, while adaptive intelligent modeling uses feedback from the results of the search to decide on future experiments.

Adaptive intelligent modeling represents a promising and important direction in code optimization. The defining motivation is to integrate all relevant information into a hybrid model which can both resolve optimization decisions and guide further information collection. The challenge is combining information from different sources that come in radically different forms. In this first proof of concept research, the forms include a general but approximate prior analytical model and empirical measurements of code samples taken directly on the system to be optimized. As mentioned in the introduction, we chose the matrix multiplication problem only because it is a well-studied domain on which our approach can be tested. Consequently, our comparison with ATLAS is designed to show the efficacy of our solution by comparing its performance with that of a well-studied system. In this chapter, and mainly as a mean to evaluate our technique in a more complex environment we extended the search range of ATLAS to also consider the L2 peak area. We did that considering MMM to be a stand-alone application in its own right. As outlined in Section 4.3, this approach may not be optimal when the MMM library is called from other linear algebra routines [1].

The algorithms in this chapter are rooted in the traditional Bayesian framework for interpreting prior knowledge and, therefore, suffer from many of its shortcomings: unrealistic assumptions about the Gaussian distribution of noise and the use of a conjugate Gaussian prior for efficiency. However, the most significant research contribution of the work reported in this chapter is to open a new direction for code optimization. The principle of adaptive intelligent modeling is to actively seek out information that can be used as evidence for refining and restructuring itself so that the optimization decisions are always the best they can be.

In matrix multiplication, the main advantage of applying adaptive intelligent modeling comes from using low-cost sample points to decide when it is worthwhile to tile for the L2 cache. In this test domain of mini-MMM optimization, our adaptive model is competitive among the three experimental approaches. More importantly, as conjectured, the approach outperforms the prior performance model and is much more efficient than the empirical optimization approach. On average the adaptive model performs 62% better than the prior performance model; compared to the empirical search of ATLAS, the adaptive model runs, on average, more than four times faster. Greater improvements may be achieved.

Chapter 5

Conclusions

In this dissertation, I introduced three frameworks for improving agents' performance with prior knowledge in the context of classification, reinforcement learning, and regression. Both empirical results and theoretical guarantees demonstrate improvement in performance due to prior knowledge. However, several important questions remain to be investigated. One unresolved problem is how to balance prior knowledge with new information the agent receives either from exploration or labeled training data. Since we never expect prior knowledge to be exactly right, we want the collected data to eventually overwhelm prior knowledge, resulting in an agent which, given enough experience, tailors its behavior more to the requirements fo the actual world. While this occurs in all of our frameworks, our ontology-based classification agent can assign too much weight to the prior, resulting in suboptimal behavior for larger data samples (see Figure 2.5.2-(4)). We have not observed this phenomenon in the other frameworks. However, there are very few formal guidelines for determining the optimal tradeoff between fitting the data and respecting prior knowledge [1]. Another challenging open problem is determining the proper form for expressing and interpreting prior knowledge. This form can range from a very weak prior which allows the domain expert to specify constraints on the kinds of behavior the world can exhibit (as in Qualitative Reinforcement Learning) to a very strong, but possibly inaccurate prior which specifies exactly how the world behaves (as in Active Reinforcement Learning and Compiler Optimization). Another alternative is utilizing prior knowledge which was not specifically designed for any particular learning algorithm (e.g., Ontologies for text categorization), in which case an important open problem is how to map this prior into the kind of representation that a learning algorithm can use. Determining which kind of prior knowledge works best for capturing experts' intuitions and which kind of prior knowledge improves performance the most is an important direction for future research.

[1] Cross-validation can be used to set the tradeoff parameter. However, in our classification experiments, cross-validation did not work well in practice.

Appendix A

Prior for Text Categorization

A.1 Convergence of the Generative/Discriminative Algorithm

Let the map $H : Z \to Z$ determine an algorithm that, given a point $\mu^{(0)}$, generates a sequence $\left\{\mu^{(t)}\right\}_{t=0}^{\infty}$ of iterates through the iteration $\mu^{(t+1)} = H(\mu^{(t)})$. The iterative algorithm in Section 2.4 generates a sequence of iterates $\mu^{(t)} = [\mu_1^{(t)}, \mu_2^{(t)}] \in Z$ by applying the following map H:

$$H = H_2 \circ H_1 : \tag{A.1.1}$$

$$\text{In step 1, } H_1([\mu_1, \mu_2]) = \arg \min_{[w,b] \in U([\mu_1,\mu_2])} \|w\|, \tag{A.1.2}$$

$$\text{with the set } U([\mu_1, \mu_2]) \text{ defined by constraints:} \tag{A.1.3}$$

$$y_i(w^T x_i + b) - 1 \geq 0, i = 1, .., m \tag{A.1.4}$$

$$c_{-1}(w, b; \mu_1, \Sigma_1) - \beta \geq 0 \tag{A.1.5}$$

$$c_1(w, b; \mu_2, \Sigma_2) - \beta \geq 0 \tag{A.1.6}$$

with the conic constraints $c_s(w, b; \mu, \Sigma) \triangleq s\left(\frac{w^T\mu+b}{\|\Sigma^{1/2}w\|}\right)$.

$$\text{In step 2, } H_2(w, b) = \arg \min_{(\mu_1,\mu_2)\in V} -(c_{-1}(w, b; \mu_1, \Sigma_1) + c_1(w, b; \mu_2, \Sigma_2)) \tag{A.1.7}$$

$$\text{with the set } V \text{ given by the constraints}$$

$$\varphi - o(\mu_1; \Omega_1, t_1) \geq 0 \tag{A.1.8}$$

$$\varphi - o(\mu_2; \Omega_2, t_2) \geq 0 \tag{A.1.9}$$

with $o(\mu; \Omega, t) \triangleq \left\|\Omega^{-1/2}(\mu - t)\right\|$.

Notice that H_1 and H_2 are functions because the minima for optimization problems (2.4.10)-(2.4.14) and (2.4.15)-(2.4.16) are unique. This is the case because Step 1 optimizes a strictly convex function on a convex set, and Step 2 optimizes a linear non-constant function on a strictly convex set.

Convergence of the objective function $\psi(\mu^{(t)}) \triangleq \min_{[w,b]\in U([\mu_1^{(t)},\mu_2^{(t)}])} \|w\|$ of the algorithm was shown in Theorem 2.4.1. Let Γ denote the set of points on which the map H does not change the value of the objective function, i.e. $\mu^* \in \Gamma \Leftrightarrow \psi(H(\mu^*)) = \psi(\mu^*)$. We will show that every accumulation point of $\{\mu^{(t)}\}$ lies in Γ. We will also show that every point $[\mu_1^*, \mu_2^*] \in \Gamma$ augmented with $[w^*, b^*] = H_1([\mu_1^*, \mu_2^*])$ is a point with no feasible descent directions for the optimization problem (2.4.5)-(2.4.9), which can be equivalently expressed as:

$$\min_{\mu_1,\mu_2,w,b} \|w\| \text{ s.t.} [\mu_1, \mu_2] \in V; [w, b] \in U([\mu_1, \mu_2]) \tag{A.1.10}$$

In order to formally state our result, we need a few concepts from the duality theory. Let a constrained optimization problem be given by

$$\min_x f(x) \text{ s.t. } c_i(x) \geq 0, i = 1, .., k \tag{A.1.11}$$

The following conditions, known as Karush-Kuhn-Tucker(KKT) conditions are necessary for x^* to be a local minimum:

Proposition A.1.1. *If x^* is a local minimum of (A.1.11), then $\exists \lambda_1, .., \lambda_k$ such that*

1. $\nabla f(x^*) = \sum_{i=1}^k \lambda_i \nabla c_i(x^*)$
2. $\lambda_i \geq 0$ *for* $\forall i \in \{1, .., k\}$
3. $c_i(x^*) \geq 0$ *for* $\forall i \in \{1, .., k\}$
4. $\lambda_i c_i(x^*) = 0$ *for* $\forall i \in \{1, .., k\}$

$\lambda_1, .., \lambda_k$ are known as Lagrange multipliers of constraints $c_1, .., c_k$.

The following well-known result states that KKT conditions are sufficient for x^* to be a point with no feasible descent directions:

Proposition A.1.2. *If $\exists \lambda_1, .., \lambda_k$ such that the following conditions are satisfied at x^*:*

1. $\nabla f(x^*) = \sum_{i=1}^k \lambda_i \nabla c_i(x^*)$
2. $\lambda_i \geq 0$ *for* $\forall i \in \{1, .., k\}$

then x^* has no feasible descent directions in the problem (A.1.11)

Proof. (sketch) We reproduce the proof given in a textbook by Fletcher [36]. The proposition is true because for any feasible direction vector s, $s^T \nabla c_i(x) \geq 0$ for $\forall x$ and for $\forall i \in \{1, .., k\}$. Hence, $s^T \nabla f(x^*) = \sum_{i=1}^{k} \lambda_i s^T \nabla c_i(x^*) \geq 0$, so s is not a descent direction. □

The following lemma characterizes the points in the set Γ:

Lemma A.1.3. *Let $\mu^* \in \Gamma$, and let $[w^*, b^*] = H_1(\mu^*)$ be the optimizer of $\psi(\mu^*)$, and let $\boldsymbol{\lambda}^* = [\lambda^*_{(A.1.4),1}, .., \lambda^*_{(A.1.4),m}, \lambda^*_{(A.1.5)}, \lambda^*_{(A.1.6)}]$ be a set of Lagrange multipliers corresponding to the constraints for the solution $[w^*, b^*]$. Define $\mu' = H(\mu^*)$, and let $[w', b']$ be the optimizer of $\psi(\mu')$. If $\mu'_2 \neq \mu^*_2$, then $\lambda^*_{(A.1.6)} = 0$ for some $\boldsymbol{\lambda}^*$. If $\mu'_1 \neq \mu^*_1$, then $\lambda^*_{(A.1.5)} = 0$ for some $\boldsymbol{\lambda}^*$. If both $\mu'_1 \neq \mu^*_1$ and $\mu'_2 \neq \mu^*_2$, then $\lambda^*_{(A.1.6)} = \lambda^*_{(A.1.5)} = 0$ for some $\boldsymbol{\lambda}^*$.*

Proof. Consider the case when

$$\mu'_2 \neq \mu^*_2 \tag{A.1.12}$$

and

$$\mu'_1 = \mu^*_1 \tag{A.1.13}$$

Since $\mu^* \in \Gamma$, $\|w'\| = \|w^*\|$. Let $\boldsymbol{\lambda}'$ be a set of Lagrange multipliers corresponding to the constraints for the solution $[w', b']$. Since w^* is still feasible for the optimization problem given by $\psi(\mu')$ (by the argument in Theorem 2.4.1) and the minimum of this problem is unique, this can only happen if

$$[w', b'] = [w^*, b^*]. \tag{A.1.14}$$

Then $[w^*, b^*]$ and $\boldsymbol{\lambda}'$ must satisfy KKT conditions for $\psi(\mu')$. (A.1.12) implies that $c_1(w^*; \mu'_2, \Sigma_2) > c_1(w^*; \mu^*_2, \Sigma_2) \geq \beta$ by the same argument as in Theorem 2.4.1, which means that, by KKT condition (4) for $\psi(\mu')$,

$$\lambda'_{(\text{A.1.6})} = 0. \tag{A.1.15}$$

Therefore, by KKT condition (1) for $\psi(\mu')$ and (A.1.15), at $[w, b, \mu_1, \mu_2] = [w^* = w', b^* = b', \mu^*_1 = \mu'_1, \mu^*_2]$

$$\begin{bmatrix} \frac{\partial \|w\|}{\partial w} \\ \frac{\partial \|w\|}{\partial b} \end{bmatrix} = \sum_{i=1}^{m} \lambda'_{(\text{A.1.4}),i} \begin{bmatrix} y_i x_i \\ y_i \end{bmatrix} + \lambda'_{(\text{A.1.5})} \begin{bmatrix} \frac{\partial c_{-1}(w, b^*; \mu^*_1, \Sigma_1)}{\partial w} \\ \frac{\partial c_{-1}(w^*, b; \mu^*_1, \Sigma_1)}{\partial b} \end{bmatrix} + \lambda'_{(\text{A.1.6})} \begin{bmatrix} \frac{\partial c_1(w, b^*; \mu^*_2, \Sigma_2)}{\partial w} \\ \frac{\partial c_1(w^*, b; \mu^*_2, \Sigma_2)}{\partial b} \end{bmatrix},$$

which means that KKT conditions (1),(2) for the optimization problem $\psi(\mu^*)$ are satisfied at the point

$[w^*, b^*]$ with $\boldsymbol{\lambda}^* = \boldsymbol{\lambda}^{'}$. KKT condition (3) is satisfied by feasibility of $[w^*, b^*]$ and KKT condition (4) is satisfied by the same condition for $\psi(\mu')$ and observations (A.1.13), (A.1.14), and (A.1.15).

The proofs for the other two cases ($\mu_2' = \mu_2^*, \mu_1' \neq \mu_1^*$ and $\mu_2' \neq \mu_2^*, \mu_1' \neq \mu_1^*$) are analogous. □

The following theorem states that the points in Γ are KKT points (i.e., points at which KKT conditions are satisfied) for the optimization problem given by (A.1.10).

Theorem A.1.4. *If $\mu^* \in \Gamma$ and let $[w^*, b^*] = H_1(\mu^*)$, then $[w^*, b^*, \mu_1^*, \mu_2^*]$ is a KKT point for the optimization problem given by (A.1.10).*

Proof. Let $\mu' = H(\mu^*)$. Just like in Lemma A.1.3, we only consider the case

$$\mu_2' \neq \mu_2^*, \tag{A.1.16}$$

$$\mu_1' = \mu_1^* \Rightarrow \lambda^*_{(\text{A.1.6})} = 0 \text{ (by Lemma A.1.3).} \tag{A.1.17}$$

(the proofs for the other two cases are similar).

By KKT conditions for $H_2(w^*, b^*)$, at $\mu_1 = \mu_1'$

$$-\frac{\partial c_{-1}(w^*, b^*; \mu_1, \Sigma_1)}{\partial \mu_1} = \lambda'_{\text{A.1.8}} \frac{\partial(-o(\mu_1; \Omega_1, t))}{\partial \mu_1} \text{ for some } \lambda'_{\text{A.1.8}} \geq 0. \tag{A.1.18}$$

By KKT conditions for $H_1(\mu^*)$ and (A.1.17), at $[w, b] = [w^*, b^*]$

$$\begin{bmatrix} \frac{\partial \|w\|}{\partial w} \\ \frac{\partial \|w\|}{\partial b} \end{bmatrix} = \sum_{i=1}^{m} \lambda^*_{(\text{A.1.4}),i} \begin{bmatrix} y_i x_i \\ y_i \end{bmatrix} + \lambda^*_{(\text{A.1.5})} \begin{bmatrix} \frac{\partial c_{-1}(w, b^*; \mu_1^*, \Sigma_1)}{\partial w} \\ \frac{\partial c_{-1}(w^*, b; \mu_1^*, \Sigma_1)}{\partial b} \end{bmatrix} \text{ for some } \begin{bmatrix} \lambda^*_{(\text{A.1.4}),1} \\ .. \\ \lambda^*_{(\text{A.1.4}),m} \\ \lambda^*_{(\text{A.1.5})} \end{bmatrix} \succeq 0. \tag{A.1.19}$$

By (A.1.16),(A.1.17),(A.1.18), and (A.1.19), at $[w, b, \mu_1, \mu_2] = [w^*, b^*, \mu_1^* = \mu_1', \mu_2^*]$

$$\begin{bmatrix} \frac{\partial \|w\|}{\partial w} \\ \frac{\partial \|w\|}{\partial b} \\ \frac{\partial \|w\|}{\partial \mu_1} \\ \frac{\partial \|w\|}{\partial \mu_2} \end{bmatrix} = \begin{bmatrix} \frac{\partial \|w\|}{\partial w} \\ 0 \\ 0 \\ 0 \end{bmatrix} = \sum_{i=1}^{m} \lambda^*_{(\text{A.1.4}),i} \begin{bmatrix} y_i x_i \\ y_i \\ 0 \\ 0 \end{bmatrix} + \lambda^*_{(\text{A.1.5})} \begin{bmatrix} \frac{\partial c_{-1}(w, b^*; \mu_1^*, \Sigma_1)}{\partial w} \\ \frac{\partial c_{-1}(w^*, b; \mu_1^*, \Sigma_1)}{\partial b} \\ \frac{\partial c_{-1}(w^*, b^*; \mu_1, \Sigma_1)}{\partial \mu_1} \\ 0 \end{bmatrix} +$$

$$\lambda'_{\text{A.1.8}} \lambda^*_{(\text{A.1.5})} \begin{bmatrix} 0 \\ 0 \\ \frac{\partial(-o(\mu_1; \Omega_1, t))}{\partial \mu_1} \\ 0 \end{bmatrix} + \lambda^*_{(\text{A.1.6})} \begin{bmatrix} \frac{\partial c_1(w, b^*; \mu_2^*, \Sigma_2)}{\partial w} \\ \frac{\partial c_1(w^*, b; \mu_2^*, \Sigma_2)}{\partial b} \\ 0 \\ \frac{\partial c_1(w^*, b^*; \mu_2, \Sigma_2)}{\partial \mu_2} \end{bmatrix} + \lambda^*_{(\text{A.1.6})} \begin{bmatrix} 0 \\ 0 \\ 0 \\ \frac{\partial(-o(\mu_2; \Omega_2, t))}{\partial \mu_2} \end{bmatrix},$$

which means that KKT conditions (1),(2) for the optimization problem (A.1.10) are satisfied at the point $[w^*, b^*, \mu_1^*, \mu_2^*]$ with $\boldsymbol{\lambda}^{''} = [\lambda^*_{(\text{A.1.4}),1}, .., \lambda^*_{(\text{A.1.4}),m}, \lambda^*_{(\text{A.1.5})}, \lambda^*_{(\text{A.1.6})}, \lambda'_{\text{A.1.8}} \lambda^*_{(\text{A.1.5})}, \lambda^*_{(\text{A.1.6})}]$. $\boldsymbol{\lambda}^{''}$ also satisfies KKT conditions (3),(4) by assumption (A.1.17) and the KKT conditions for H_1 and H_2. □

In order to prove convergence properties of the iterates $\mu^{(t)}$, we use the following theorem due to Zangwill [84]:

Theorem A.1.5. *Let the map $H : Z \to Z$ determine an iterative algorithm via $\mu^{(t+1)} = H(\mu^{(t)})$, let $\psi(\mu)$ denote the objective function, and let Γ be the set of points on which the map H does not change the value of the objective function, i.e. $\mu \in \Gamma \Leftrightarrow \psi(H(\mu)) = \psi(\mu)$. Suppose*

1. *H is uniformly compact on Z, i.e. there is a compact subset $Z_0 \subseteq Z$ such that $H(\mu) \in Z_0$ for $\forall \mu \in Z$.*
2. *H is strictly monotonic on $Z - \Gamma$, i.e. $\psi(H(\mu)) < \psi(\mu)$.*
3. *H is closed on $Z - \Gamma$, i.e. if $w_i \to w$ and $H(w_i) \to \xi$, then $\xi = H(w)$.*

Then the accumulation points of the sequence of $\mu^{(t)}$ lie in Γ.

The following proposition shows that minimization of a continuous function on a feasible set which is a continuous map of the function's argument forms a closed function.

Proposition A.1.6. *Given*

1. *a real-valued continuous function f on $A \times B$,*
2. *a point-to-set map $U : A \to 2^B$ continuous with respect to the Hausdorff metric:*[1] *$dist(X, Y) \triangleq \max(d(X, Y), d(Y, X))$, where $d(X, Y) \triangleq \max_{x \in X} \min_{y \in Y} \|x - y\|$,*

[1] A point-to-set map $U(a)$ maps a point a to a set of points. $U(a)$ is continuous with respect to a distance metric *dist* iff $a^{(t)} \to a$ implies $dist(U(a^{(t)}), U(a)) \to 0$.

define the function $F : A \to B$ *by*

$$F(a) = \arg\min_{b' \in U(a)} f(a, b') = \{b : f(a, b) < f(a, b') \text{ for } \forall b' \in U(a)\},$$

assuming the minimum exists and is unique. Then, the function F *is closed at* a*.*

Proof. This proof is a minor modification of the one given by Gunawardana and Byrne [42]. Let $\{a^{(t)}\}$ be a sequence in A such that

$$a^{(t)} \to a, F(a^{(t)}) \to b \tag{A.1.20}$$

The function F is closed at a if $F(a) = b$. Suppose this is not the case, i.e. $b \neq F(a) = \arg\min_{b' \in U(a)} f(a, b')$. Therefore,

$$\exists \hat{b} = \arg\min_{b' \in U(a)} f(b') \text{ such that } f(a, b) > f(a, \hat{b}) \tag{A.1.21}$$

By continuity of $f(\cdot, \cdot)$ and (A.1.20),

$$f(a^{(t)}, F(a^{(t)})) \to f(a, b) \tag{A.1.22}$$

By continuity of $U(\cdot)$ and (A.1.20),

$$dist(U(a^{(t)}), U(a)) \to 0 \Rightarrow \exists \hat{b}^{(t)} \to \hat{b} \text{ and } \hat{b}^{(t)} \in U(a^t), \text{ for } \forall t. \tag{A.1.23}$$

(A.1.22), (A.1.23), and (A.1.21) imply that

$$\exists K \text{ such that } f(a^{(t)}, F(a^{(t)})) > f(a^{(t)}, \hat{b}^{(t)}), \text{ for } \forall t > K \tag{A.1.24}$$

which is a contradiction since by assumption, $F(a^{(t)}) = \arg\min_{b' \in U(a^t)} f(b')$ and by (A.1.24), $\hat{b}^{(t)} \in U(a^{(t)})$. □

Proposition A.1.7. *The function* H *defined by (A.1.1)-(A.1.7) is closed.*

Proof. Let $\{\mu^{(t)}\}$ be a sequence such that $\mu^{(t)} \to \mu^*$. Since all the iterates $\mu^{(t)}$ lie in the closed feasible region bounded by constraints (2.4.6)-(2.4.9) and the boundary of $U(\mu)$ is piecewise linear in μ, the boundary of $U(\mu^{(t)})$ converges uniformly to the boundary of $U(\mu^*)$ as $\mu^{(t)} \to \mu^*$, which implies that the Hausdorff distance between the boundaries converges to zero. Since the Hausdorff distance between convex sets is equal to the Hausdorff distance between their boundaries, $dist(U(\mu^{(t)}), U(\mu^*))$ also converges to zero. Hence, proposition A.1.6 implies that H_1 is closed. The same proposition implies that H_2 is closed. A composition of closed functions is closed, hence H is closed. □

We now prove the main result of this section:

Theorem 2.4.2. *Let H be the function defined by (A.1.1)-(A.1.7) which determines the generative/discriminative algorithm via $\mu^{(t+1)} = H(\mu^{(t)})$. Then accumulation points μ^* of the sequence $\mu^{(t)}$ augmented with $[w^*, b^*] = H_1(\mu^*)$ have no feasible descent directions for the original optimization problem given by (2.4.5)-(2.4.9).*

Proof. The proof is by verifying that H satisfies the properties of Theorem A.1.5. Closedness of H was shown in Proposition A.1.7. Strict monotonicity of $\psi(\mu^{(t)})$ was shown in Theorem 2.4.1. Since all the iterates $\mu^{(t)}$ are in the closed feasible region bounded by constraints (2.4.6)-(2.4.9), H is uniformly compact on Z. Since all the accumulation points μ^* lie in Γ, they are KKT points of the original optimization problem by Theorem A.1.4, and, therefore, have no feasible descent directions by Proposition A.1.2. □

A.2 Generalization of the Generative/Discriminative Classifier

We need a few auxiliary results before proving Theorem 2.6.2. The first proposition bounds the angle of rotation between two vectors w_1, w_2 and the distance between them if the angle of rotation between each of these vectors and some reference vector v is sufficiently small:

Proposition A.2.1. *Let $\|w_1\| = \|w_2\| = \|v\| = 1$. If $w_1^T v \geq \alpha \geq 0$ and $w_2^T v \geq \alpha \geq 0$, then*

1. $w_1^T w_2 \geq 2\alpha^2 - 1$
2. $\|w_1 - w_2\| \leq 2\sqrt{(1-\alpha^2)}$

Proof.

1. By the triangle inequality, $\arccos(w_1^T w_2) \leq \arccos(w_1^T v) + \arccos(w_2^T v) \leq 2\arccos(\alpha)$ (since the angle between two vectors is a distance measure). Taking cosines of both sides and using trigonometric equalities yields $w_1^T w_2 \geq 2\alpha^2 - 1$.
2. Expand $\|w_1 - w_2\|^2 = \|w_1\|^2 + \|w_2\|^2 - 2w_1^T w_2 = 2(1 - w_1^T w_2)$. Since $w_1^T w_2 \geq 2\alpha^2 - 1$ from part 1, $\|w_1 - w_2\|^2 \leq 4(1-\alpha^2)$.

□

The next proposition bounds the angle of rotation between two vectors t and μ if they are not too far away from each other as measured by the L_2-norm distance:

Proposition A.2.2. *Let $\|t\| = \nu$, $\|\mu - t\| \leq \tau$. Then $\frac{t^T \mu}{\|t\|\|\mu\|} \geq \frac{\nu^2 - \tau^2}{\nu(\nu+\tau)}$.*

Proof. Expanding $\|\mu - t\|^2 = \|t\|^2 + \|\mu\|^2 - 2t^t\mu$ and using $\|\mu - t\|^2 \leq \tau^2$, we get $\frac{t^T\mu}{\|t\|\|\mu\|} \geq \frac{1}{2}(\frac{\|t\|}{\|\mu\|} + \frac{\|\mu\|}{\|t\|} - \frac{\tau^2}{\|t\|\|\mu\|})$. We now use the triangle inequality $\nu - \tau \leq \|t\| - \|\mu - t\| \leq \|\mu\| \leq \|t\| + \|\mu - t\| \leq \nu + \tau$ and simplify. □

The following proposition will be used to bound the angle of rotation between the normal w of the separating hyperplane and the mean vector t of the hyper-prior distribution:

Proposition A.2.3. *Let* $\frac{w^T\mu}{\|w\|\|\mu\|} \geq \beta \geq 0$ *and* $\|\mu - t\| \leq \varphi \leq \|t\|$. *Then* $\frac{w^Tt}{\|w\|\|t\|} \geq (2\alpha^2 - 1)$, *where* $\alpha = \min(\beta, \frac{\|t\|^2 - \varphi^2}{\|t\|(\varphi + \|t\|)})$.

Proof. Follows directly from Propositions A.2.1 (part 1) and A.2.2. □

We now prove Theorem 2.6.2, which relies on parts of the well-known proof of the fat-shattering dimension bound for large margin classifiers derived by Taylor and Bartlett [71].

Theorem 2.6.2. *Let F be the class of a-priori constrained functions defined by 2.6.1, and let $\lambda_{min}(P)$ and $\lambda_{max}(P)$ denote the minimum and maximum eigenvalues of matrix P, respectively. If a set of points S is γ-shattered by F, then $|S| \leq \frac{4R^2(\alpha^2(1-\alpha^2))}{\gamma^2}$, where $\alpha = \max(\alpha_1, \alpha_2)$ with $\alpha_1 = \min(\frac{\lambda_{min}(\Sigma_1)\beta}{\|\mu_2\|}, \frac{\|t_2\|^2 - (\lambda_{max}(\Omega_2)\varphi)^2}{\|t_2\|(\lambda_{max}(\Omega_2)\varphi)^2 + \|t_2\|)})$ and $\alpha_2 = \min(\frac{\lambda_{min}(\Sigma_1)\beta}{\|\mu_1\|}, \frac{\|t_1\|^2 - (\lambda_{max}(\Omega_1)\varphi)^2}{\|t_1\|((\lambda_{max}(\Omega_1)\varphi)^2 + \|t_1\|)})$, assuming that $\beta \geq 0$, $\|t_i\| \geq \|t_i - \mu_i\|$, and $\alpha_i \geq \frac{1}{\sqrt{2}}, i = 1, 2$.*

Proof. First, we use the inequality $\lambda_{min}(P)\|w\| \leq \left\|P^{1/2}w\right\| \leq \lambda_{max}(P)\|w\|$ to relax the constraints

$$\frac{w^T\mu_2}{\left\|\Sigma_2^{1/2}w\right\|} \geq \beta \Rightarrow \frac{w^T\mu_2}{\|w\|} \geq \lambda_{min}(\Sigma_2)\beta \tag{A.2.1}$$

$$\left\|\Omega_2^{-1/2}(\mu_2 - t_2)\right\| \leq \varphi \Rightarrow \|\mu_2 - t_2\| \leq \frac{\varphi}{\lambda_{min}(\Omega_2^{-1})} = \varphi\lambda_{max}(\Omega_2). \tag{A.2.2}$$

The constraints imposed by the second prior $\frac{-w^T\mu_1}{\left\|\Sigma_2^{1/2}w\right\|} \geq \beta, \left\|\Omega_1^{-1/2}(\mu_1 - t_1)\right\| \leq \varphi$ are relaxed in a similar fashion to produce:

$$\frac{w^T(-\mu_1)}{\|w\|} \geq \lambda_{min}(\Sigma_1)\beta \tag{A.2.3}$$

$$\|\mu_1 - t_1\| \leq \varphi\lambda_{max}(\Omega_1) \tag{A.2.4}$$

Now, we show that if the assumptions made in the statement of the theorem hold, then every subset $S_o \subseteq S$ satisfies $\|\sum S_o - \sum(S - S_0)\| \leq \frac{4R^2(\alpha^2(1-\alpha^2)}{\gamma^2}$.

Assume that S is γ-shattered by F. The argument used by Taylor and Bartlett [71] in Lemma 1.2 shows that, by the definition of fat-shattering, there exists a vector w_1 such that

$$w_1(\sum S_o - \sum(S - S_0)) \geq |S|\,\gamma. \tag{A.2.5}$$

Similarly (reversing the labeling of S_0 and $S_1 - S_0$), there exists a vector w_2 such that

$$w_2(\sum(S - S_0) - \sum S_o) \geq |S|\,\gamma. \tag{A.2.6}$$

Hence, $(w_1 - w_2)(\sum S_o - \sum(S - S_0)) \geq 2\,|S|\,\gamma$, which, by Cauchy-Schwartz inequality, implies that

$$\|w_1 - w_2\| \geq \frac{2\,|S|\,\gamma}{\|\sum S_o - \sum(S - S_0)\|} \tag{A.2.7}$$

The constraints on the classifier represented in A.2.1 and A.2.2 imply by Proposition A.2.3 that $\frac{w_1^T t_2}{\|w_1\|\|t_2\|} \geq (2\alpha_1^2 - 1)$ and $\frac{w_2^T t_2}{\|w_2\|\|t_2\|} \geq (2\alpha_2^2 - 1)$. Now, applying Proposition A.2.1 (part 2) and simplifying, we get

$$\|w_1 - w_2\| \leq 4\sqrt{\alpha_1^2(1 - \alpha_1^2)}. \tag{A.2.8}$$

Applying the same analysis to the constraints A.2.3 and A.2.4, we get

$$\|w_1 - w_2\| \leq 4\sqrt{\alpha_2^2(1 - \alpha_2^2)}. \tag{A.2.9}$$

Combining A.2.7, A.2.8, and A.2.9, we get

$$\left\|\sum S_o - \sum(S - S_0)\right\| \geq \frac{|S|\,\gamma}{2\sqrt{\alpha^2(1 - \alpha^2)}} \tag{A.2.10}$$

with α as defined in the statement of the theorem.

Taylor and Bartlett's [71] Lemma 1.3 proves, using the probabilistic method, that some $S_o \subseteq S$ satisfies

$$\left\|\sum S_o - \sum(S - S_0)\right\| \leq \sqrt{|S|}R. \tag{A.2.11}$$

Combining A.2.10 and A.2.11 yields $|S| \leq \frac{4R^2(\alpha^2(1-\alpha^2))}{\gamma^2}$. □

Appendix B

Qualitative Reinforcement Learning

B.1 Preliminaries

The following proposition verifies that $\preceq$ is a total order, which means that a maximum is well-defined for any finite set of vectors. We take the $\otimes$ operator to be this maximum: $\otimes_a^{(s)} f(s,a) = max_a^{\prec} f(s,a)$.

Theorem B.1.1. *$\prec$ is a strict partial order (i.e., an irrefexive transitive relation). $\preceq$ is a total order (i.e., complete reflexive antisymmetric transitive relation).*

Proof. $\prec$ is a strict partial order:

1. Irreflexivity. Let $U = V$. Then $f^*(U,V)$ does not exist, so $U \not\prec V$.
2. Transitivity. Given $U \prec V \prec Z$, we have to show that $U \prec Z$. Let $a = f^*(U,V)$, $b = f^*(V,Z)$, $c = f^*(U,Z)$. Assume that $Z \prec U$. Then, wlog, a, b, c exist and $a \leq b \leq c$ Since $V \prec Z \prec U$, $V_a \leq Z_a \leq U_a$. However, since $U \prec V$, $U_a < V_a$, a contradiction. Therefore, either $U = Z$ or $U \prec Z$. However, $U = Z$ and $U \prec V$ implies $Z \prec V$, a contradiction. Therefore, $U \prec Z$.

Since $\preceq$ is complete, and it is a strict partial order augmented with an identity function, it is a total order. □

Note that the policy evaluation equation $V_\pi^{t+1}(s) = [KV_\pi^t](s, \pi(s))$ (3.1.2) can be equivalently expressed as

$$V_{j,\pi}^t = \begin{cases} r(s,\pi(s)), j = 0 \\ E_{P(s'|s,\pi(s))} V_{j-1,\pi}^{t-1}(s'), j \geq 1 \end{cases} \tag{B.1.1}$$

where $E_P V(x) = \sum_x P(x)V(x)$ is the expected value of V, or, equivalently, in matrix notation:

$$V_{j,\pi}^t = \begin{cases} R_\pi, j = 0 \\ P_\pi V_{j-1}^{t-1}, j \geq 1 \end{cases} \tag{B.1.2}$$

where P_π is the $|S|$x$|S|$ transition matrix for policy π, $R_\pi = [r(s_1,\pi(s_1)), .., r(s_{|S|},\pi(s_{|S|}))]$ is the reward

vector, and $V^t_{j,\pi} = [V^t_{j,\pi}(s_1), .., V^t_{j,\pi}(s_{|S|})]$ is the j-th component of the value vector at iteration t of policy iteration.

We now prove a proposition which states that, in t-th iteration of policy evaluation, all components $0 \leq j \leq t$ of the value vector $V^t_\pi(s)$ represent expected rewards received after following π from s for j steps:

Proposition B.1.2. $V^t_{j,\pi} = \begin{cases} P^j_\pi R, 0 \leq j \leq t \\ P^j_\pi V^0_{j-1}, j \geq t+1 \end{cases}$ *, where V^0_{j-1} is the initial value function and we use the convention that P^0_π is the identity matrix.*

Proof. The proof is by induction on t: The base case ($t = 1$) follows directly from (B.1.2). For the inductive step, we have $V^{t+1}_j = \begin{cases} R, j = 0 \\ P_\pi P^{j-1}_\pi R, 1 \leq j \leq t \\ P_\pi P^{j-1}_\pi V^0_{j-1}, j \geq t+1 \end{cases}$

□

We now prove convergence of policy evaluation:

Theorem B.1.3. *For any fixed policy π, there is a unique optimal value function V^*_π which satisfies $V^*_\pi = KV^*_\pi$, and policy evaluation converges to V^*_π, in the distance metric defined by $d(U,V) = |\sum_{j=0}^{N} U_j \alpha^j - \sum_{j=0}^{N} V_j \alpha^j|$, any $\alpha \in (0,1)$.*

Proof. First notice that, since U and V are componentwise non-negative, d is a true distance metric. By Proposition B.1.2, $\sum_{j=0}^{N} V^t_{j,\pi} \alpha^j = \sum_{j=0}^{t} \alpha^j P^j_\pi R + \sum_{j=t+1}^{N} \alpha^j P^j_\pi V^0_{j-1}$, which is exactly the value function at iteration t of conventional MDP policy iteration with discount factor α and initial policy $V^0 = \sum_{j=0}^{N} V^0_j$. Since conventional MDP policy evaluation converges, so does myopic MDP policy evaluation. Moreover, V^*_π must be the only fixed point of the myopic KV operator since if that were not the case, conventional MDP operator $K_0 V$ would also have multiple fixed points. □

Convergence of policy iteration follows from the following proposition, which establishes equivalence between myopic policy iteration and conventional MDP policy iteration with a low discount factor α:

Lemma B.1.4. *Let myopic policy iteration pick actions $a^t(s)$ in iterations $t = 1, 2, ...$ after applying the HV_π^t operator. Then there is a value of $\xi \in (0,1)$ such that conventional policy iteration with any discount factor $\alpha \in (0, \xi)$ picks the same actions in corresponding iterations after applying $H_0 V_\pi^t$.*

Proof.

$$\text{Let } \lambda = \max_{\pi,s,a_1,a_2}{}^{+} f^*([KV_\pi^*](s,a_1)), [KV_\pi^*](s,a_2)),$$

where V_π^* is the optimal value function under policy π, and $\max^+$ is the maximum is taken over the domain on which f^* exists, i.e.

$$max^+ f(x) = \begin{cases} \max_{x:f(x)<\infty} f(x) & \text{, if } \exists a : f(a) < \infty \\ 0 & \text{, otherwise} \end{cases}$$

$$\text{Let } \epsilon = \min_{z \in \{0,1,..,\lambda\}} | \sum_{s'} ([P_\pi^z](s')(P(s'|s,a_1) - P(s'|s,a_2))|$$

$$\text{Let } M = \max_{s,a} r(s,a) \text{ be the upper bound on rewards.}$$

Given that in step t of myopic policy iteration, action a^* is chosen in state s, we will show that conventional policy iteration also chooses a^* in s if its discount factor is $\alpha < \xi = \frac{\epsilon}{2M+\epsilon}$. Pick another action a' and let $b = f^*([KV_\pi^*](s,a^*), [KV_\pi^*](s,a'))$. Since a^* is preferred to a', $[K_0 V_\pi^*](s,a^*) - [K_0 V_\pi^*](s,a') = \alpha^b \sum_{s'} ([P_\pi^b R](s')(P(s'|s,a^*) - P(s'|s,a'))) + \sum_{j=b+1}^{N} \alpha^j \sum_{s'} ([P_\pi^j R](s')(P(s'|s,a^*) - P(s'|s,a'))) \geq \alpha^b \epsilon - 2M\frac{\alpha^{b+1}}{1-\alpha} > 0$, with steps justified, in order, by Proposition B.1.2 and Definition 3.1.1, geometric series formula ($\sum_{i=n}^{N} \alpha^i < \sum_{i=n}^{\infty} \alpha^i = \frac{\alpha^n}{1-\alpha}$) and bounded rewards, and upper bound on α. □

Since conventional MDP policy iteration with discount factor α converges to the unique fixed point, we have proven the following:

Theorem B.1.5. *For the Myopic MDP, there is a unique optimal value function V^* which satisfies the myopic Bellman's equation. Policy iteration converges to V^* in the metric defined by $d(U,V) = | \sum_{j=0}^{N} U_j \alpha^j -$*

N

$\sum$ $V_j \alpha^j|$, *for any* $\alpha \in (0, \xi)$, *where* $\xi \in (0, 1)$ *is some small value which depends on the rewards and*

$j = 0$

transition probabilities of the MDP.

B.2 Convergence of Qualitative Policy Iteration

We note the following monotonicity property of stochastic dominance:

Proposition B.2.1. *For any nonnegative monotonically increasing function* $V(x \in G)$ *with respect to order* O *on* G, $E_{P_1} V(<) \leq E_{P_2} V$ *if* P_2 *(strictly) stochastically dominates* P_1 *with respect to* O, *where* $E_P V = \sum_x P(x)V(x)$ *is the expected value of* V.

Proof. We give a proof a more general statement which does not require that $P(x)$ sum to one. Monotonicity of stochastic dominance is a well-known consequence of Definition 3.1.3 [67] (and can also be shown to be true for improper probabilities by an argument similar to the one below). *Strict* monotonicity of *strict* stochastic dominance can be demonstrated by induction on $|G|$. The base case ($|G| = 1$) is true since $P_1(x_1) > P_2(x_1) \geq 0$ and $V(x_1) \geq 0$ implies $P_1(x_1)V(x_1) > P_2(x_1)V(x_1)$. For the inductive step, assume that $U(x_1) > U(x_2) > .. > U(x_k) \geq 0$ and P_1' strictly stochastically dominates P_2' implies that $E_{P_1'} U > E_{P_2'} U$. If $V(x_1) > .. > V(x_{k+1}) \geq 0$ and P_1 stochastically dominates P_2 and

$$\exists l \in \{1, .., k+1\} : \sum_{i=1}^{l} P_1(x_i) > \sum_{i=2}^{l} P_2(x_i), \text{ then}$$

$$E_{P_1} V = \sum_{i=1}^{k+1} P_1(x_i)V(x_i) =$$

$$\sum_{i=1}^{k} P_1(x_i)(V(x_i) - V(x_{k+1})) + \sum_{i=1}^{k+1} P_1(x_i)V(x_{k+1}) =$$

$$E_{P_1'}(V - V(x_{k+1})) + \sum_{i=1}^{k+1} P_1(x_i)V(x_{k+1}) >$$

$$E_{P_2'}(V - V(x_{k+1})) + \sum_{i=1}^{k+1} P_2(x_i)V(x_{k+1}) = E_{P_2} V$$

where the last inequality follows from the inductive hypothesis if $l \in \{1, ..k\}$ and monotonicity of stochastic dominance if $l = k + 1$.

□

We will prove a theorem which states that, when the qualitative policy iteration algorithm terminates, the optimal policy for any quantitative MDP consistent with the qualitative domain theory is contained in the returned candidate set of policies Π. First, we need the following Lemma:

Lemma B.2.2. *If, for some fixed policy π, qualitative policy evaluation is executed in parallel with myopic policy evaluation on any quantitative MDP consistent with the domain theory, the ordering of values $Step\_Order^j$ in iteration j of qualitative policy evaluation is consistent with the ordering of values $V^j_{j,\pi}$ of myopic policy evaluation in iteration j (with $V^0_\pi = 0$).*

Proof. This can be seen by induction on j, with the base case ($j = 0$) given by the ordering of rewards. For the inductive step, assume that $Step\_Order^j$ corresponds to the ordering of $V^j_{j,\pi}$. Consider the case when Oracle($Step\_Order^j, s_1, a_1, s_2, a_2$) returns '>'. This means that $P_\pi(s'|s_1)$ strictly stochastically dominates $P_\pi(s'|s_2)$ according to $Step\_Order^j$. By Equation (B.1.1), $V^j_{j,\pi}(s) = E_{P(s'|s,\pi(s))}V^{j-1}_{j-1,\pi}(s')$ for $j \geq 1$. Inductive assumption combined with this fact implies that $V^j_{j,\pi}(s_1) > V^j_{j,\pi}(s_2)$ by Proposition B.2.1. The proof for the cases when the Oracle returns '<' and '=' is analogous. '?', by definition, is consistent with any ordering. □

Theorem B.2.3. *If qualitative policy iteration is executed in parallel with myopic policy iteration on any quantitative MDP consistent with the domain theory, at the end of iteration t, the candidate policy set Π contains the policy returned by the conventional policy iteration algorithm at the end of t.*

Proof. The proof is by induction on t. The base case is true since both qualitative policy iteration and myopic policy iteration start out with the same policy π. Consider iteration t of policy iteration (i.e., policy evaluation followed by policy improvement). For a fixed policy π, Order in iteration j of policy evaluation is consistent with the ordering of values imposed by $\prec$ in myopic policy evaluation for any quantitative MDP consistent with the domain theory. This can be proven by induction on j, with the base case given by the ordering of rewards, and the inductive step following directly from Lemmas B.2.2 and 3.1.4. For a set of policies Π, Order is maintained to be consistent with every policy $\pi \in \Pi$. Since by Theorem B.1.3 and Lemma B.1.4, qualitative policy iteration is equivalent to conventional MDP policy iteration with small enough discount factor α, Order is also consistent with the ordering of values of this conventional MDP. Therefore, by Proposition B.2.1, policy improvement keeps all the policies which are not strictly stochastically dominated with respect to Order, and consistency of Order implies that it does not eliminate the policy selected by myopic policy improvement. □

Thus, the set of candidate policies returned by the algorithm on termination is guaranteed to contain the optimal policy.

B.3 Convergence Rate of Qualitative Policy Iteration

In this section, we will prove that this set of policies is independent of the choice of the initial policy. We will also bound the running time of the algorithm. In order to do so, we will need the following definitions:

Definition B.3.1. Let $T_\pi(s,a)$ denote a possible path through the state space imposed by performing action a in state s and following policy π afterwards. More precisely, $T_\pi(s,a) = \{s^0 = s, \pi(s^0) = a, .., s^j, \pi(s^j), ...\}$, where $s^{j+1} \in Next(s^j, \pi(s^j)),$ for $\forall j = 0, ...$

The following defines the reach of a state s, the minimum number of steps an agent has to make to ensure that it can distinguish any two paths from s from each other by encountering different rewards along the paths:

Definition B.3.2.

$$reach(s) = \max_{T_{\pi_1}, T_{\pi_2}}{}^{+} \min_{t: r(s_1^t, a_1^t) \neq r(s_2^t, a_2^t)} t$$

where $\max^+$ is defined, as in Lemma B.1.4, to be a maximum of a function over the domain where that function is finite.

Let $Reach_K$ denote the set of states such that $s \in Reach_K \Leftrightarrow reach(s) \leq K$.

We are now ready to state a theorem which bounds the number of iterations of qualitative policy iteration executed before the candidate policy set $\Pi(s)$ stops changing for every state s as a function of its reach:

Theorem B.3.3. *After K steps of policy iteration:*

- *for any two states s_1, $s_2 \in Reach_K$, $Order(s_1, s_2)$ stops changing.*
- *$\Pi(s)$ stops changing for every state $s \in Reach_K$.*
- *$\Pi(s)$ is not affected by the setting of the initial policy for every state $s \in Reach_K$*

Proof. The proof is by induction on K. In the base case ($K = 1$), if $s \in Reach_1$, then for any two actions a_1 and a_2, the rewards $r(s, a_1)$ and $r(s, a_2)$ are either different or, for any policy π, the rewards received for performing a_1 in s and following π afterwards are equivalent at every time step to the rewards received for performing a_2 in s and following π afterwards. Thus, independently of the initial policy, the optimal action at s is an action which maximimizes the immediate reward. Qualitative policy iteration chooses such maximally rewarding actions at the end of the first iteration. Since the ordering on the values of the states $s \in Reach_1$ is determined by the ordering of immediate rewards, this ordering is also fixed at the end of the first iteration. For the inductive step, any action in a state $s \in Reach_{K+1}$ must lead to

a state $s' \in Reach_K$. Since the ordering on the next states $s' \in \cup_{a \in A} Next(s,a) \subseteq Reach_K$ is fixed by the inductive hypothesis and the policy improvement step of the algorithm in Figure 3.1.2 calculates $\Pi(s)$ as a deterministic function of the ordering on s', $\Pi(s)$ is fixed after K steps. Fixing $\Pi(s)$, in turn, fixes the ordering on states $s_1, s_2 \in Reach_{K+1}$ since policy evaluation computes $Order(s_1, s_2)$ as a deterministic function of $\Pi(s)$ and the ordering on next states $s' \in \cup_{a \in \Pi(s)} Next(s,a)$. Since the ordering on the next states $s' \in \cup_{a \in A} Next(s,a)$ is not affected by the initial policy by the inductive hypothesis and the policy improvement step of the algorithm is also independent of the initial policy, $\Pi(s)$ is not affected by the initial policy for $s \in Reach_{K+1}$. □

Since the maximum (non-inifinite) reach of any state is $|S|$, the size the state space, the number of iterations of policy iteration before convergence is $O(|S|)$. An argument similar to the one in Theorem B.3.3 also shows that the number of iterations of qualitative policy evaluation before convergence is $O(|S|)$. Since each iteration of qualitative policy evaluation cycles in the worst case through all the pairs of [state, action] pairs, it performs $O(|S|^2|A|^2)$ operations. Putting all of this together, we get that the running time of qualitative policy iteration is $O(|S|^4|A|^2)$, polynomial in the size of the state/action space. This is in contrast with quantitative policy iteration, for which only pseudo-polynomial upper bounds on the running time are known (see the discussion in [53]).

B.4 Mountain Car and Cart-Pole Dynamics

In this section, we show how mountain car and cart-pole dynamics can be expressed in terms of invertible or constant functions $T_i(F; z^t)$ of F.

In the mountain car problem, $z^t = \begin{bmatrix} x_t \\ \dot{x}_t \end{bmatrix}$ and $z^{t+1} = \begin{bmatrix} x_t + \dot{x}_{t+1} \\ \dot{x}_t + Fa_t - Gcos(3x_t) \end{bmatrix} = \begin{bmatrix} T_1(z^t) \\ T_2(z^t) \end{bmatrix}$ where $T_i(z^t) = Q_i(z^t) + U_i(z^t)F, i = 1, 2$ with $Q_1(z^t) = x_t + Q_2(z^t), Q_2(z^t) = \dot{x}_{t+1} - Gcos(3x_t), U_1(z^t) = U_2(z^t) = a_t$. $T_i(F; z^t)$ is invertible (linear) when $a_t = \pm 1$ and constant when $a_t = 0$.

In the cart-pole problem, $z^t = \begin{bmatrix} \dot{h}_t \\ \dot{\theta}_t \\ \theta_t \end{bmatrix}$

Let $\ddot{\theta}_t = Q_4(z^t) + U_4(z^t)F$, where

$Q_4(z^t) = \frac{Gsin\theta_t - (m_p l \dot{\theta}_t^2 cos\theta_t sin\theta_t)/(m_c+m_p)}{l[4/3 - m_p * cos^2(\theta_t)/(m_c+m_p)]}$, $U_4(z^t) = -\frac{a_t cos(\theta_t)}{l(m_c+m_p)[4/3 - m_p * cos^2(\theta_t)/(m_c+m_p)]}$

and $\ddot{h}_t = Q_5(z^t) + U_5(z^t)F$, where

$U_5(z^t) = \frac{a_t - U_4(z^t) m_p l cos\theta_t}{m_c + m_p}$, $Q_5(z^t) = \frac{Q_4(z^t) m_p l cos\theta_t}{m_c + m_p}$, and transition dynamics are given by $z^{t+1} = \begin{bmatrix} \dot{h}_t + \tau \ddot{h}_t \\ \dot{\theta}_t + \tau \ddot{\theta}_t \\ \theta_t + \tau \dot{\theta}_t \end{bmatrix} =$

$\begin{bmatrix} T_1(F; z^t) \\ T_2(F; z^t) \\ T_3(z^t) \end{bmatrix}$ where $T_i(z^t) = Q_i(z^t) + U_i(z^t)F, i = 1, 2$ with $Q_1(z^t) = \dot{h}_t + \tau Q_5(z^t)$, $Q_2(z^t) = \dot{\theta}_t + \tau Q_4(z^t)$, $U_1(z^t) = \tau U_5(z^t)$, $U_2(z^t) = \tau U_4(z^t)$, and $T_3(z^t) = \theta_t + \tau \dot{\theta}_t$.

$T_i(F; z^t)$ is invertible (linear) for $i = 1, 2$ since $a_t = \pm 1$ and the chosen constants ensure that $U_1(z^t)$ and $U_2(z^t)$ are bounded away from zero. $T_3(z^t)$ is constant in F.

Appendix C

Active Reinforcement Learning

C.1 Convergence of Active Reinforcement Learning

In this section, we prove the results on convergence of active reinforcement learning stated in Chapter 3.2.

Lemma C.1.1. *For a given policy π and a [state,action,next state] tuple $[s, a, s'] : s' \in \overline{Next(s,a)}$, $\frac{\partial V}{\partial \overline{T}(s'|s,a)} \triangleq 0$ for $a \neq \pi(s)$. For $a = \pi(s)$, $\frac{\partial V}{\partial \overline{T}(s'|s,a)} = \frac{\alpha}{\epsilon}(I - \alpha P_0^\pi)^{-1} L V_0$, where*

$$L(s_j|s_i, \pi(s_i)) \triangleq \begin{cases} \epsilon, & \text{if } s_j = s' \\ -\epsilon, & \text{if } s_j \in \underline{Next}(s,a) \\ 0, & \text{otherwise} \end{cases}$$

Proof. The proof is taken from [17] and is given here for completeness. Suppose we want to calculate the directional derivative of V at P_0^π in the direction L. Define $P_\delta^\pi \triangleq P_0^\pi + \delta L$. By the Bellman equation, $V_0 = \alpha P_0^\pi V + R_0$ and $V_\delta = \alpha P_\delta^\pi V_\delta + R_0$. Then $V_\delta - V_0 = \alpha(P_\delta^\pi - P_0^\pi)V_0 + \alpha P_\delta^\pi (V_\delta - V_0) \Rightarrow V_\delta - V_0 = \alpha(I - \alpha P_\delta^\pi)^{-1}(P_\delta^\pi - P_0^\pi)V_0 = \alpha(I - \alpha P_\delta^\pi)^{-1}(\delta L)V_0$. Then $\nabla_L V = \frac{1}{\|L\|} \lim_{\delta \to 0} \frac{V_\delta - V_0}{\delta} = \frac{\alpha}{\|L\|}(I - \alpha P_0^\pi)^{-1} L V_0$. □

Theorem C.1.2. *Suppose that for every T and every policy π, $U^\pi(T)$ is a linear function of T. Then, for any initial point T_0', the sequence $\{T_i'\}$ generated by the variant of Newton's method in Section 3 converges to a point T_∞' on the boundary of C in a finite number of steps.*

Proof. Assume that T_i' is outside of C. By this assumption and the definition of optimal policies, $U^{\Pi_{T_i'}}(T_i') > U^{\Pi(T_0)}(T_i')$ and $U^{\Pi_{T_0'}}(T_0') \geq U^{\Pi_{T_i}}(T_0')$. Therefore, by Intermediate Value Theorem, there is a point $U \neq T_i$ on the line connecting T_i and T_0 where the value functions of the two policies $\Pi_{T_i'}$ and Π_{T_0} are equal. Since the Taylor approximation in Steps 2 and 4 of the active RL algorithm is exact and the probability simplex is convex, U will be in the feasible region of the program in Step 6. Therefore, $\left\|T_{i+1}' - T_0\right\| \leq \|U - T_0\| < \|T_i' - T_0\|$. Strict monotonicity of $\|T_i' - T_0\|$ implies that the same $\Pi_{T_i'}$ can never be picked in two different iterations i of the algorithm in Step 4. This is true because a policy for a linear value function uniquely determines the solution T_{i+1}' of the program in Step 6. Convergence in a finite number of steps follows since the number of policies is finite, and the limit point is on the boundary of C by construction. □

For dag-structured MDPs such that the maximum number of $Next$ states for any state/action pair is two, the following proposition states that Newton's method converges to the optimal boundary point.

Proposition C.1.3. *If $U^{\pi}(T)$ is a linear function of a single variable $\overline{T}(\cdot|s,a)$, then our variant of Newton's method discovers the boundary point closest to T_0 after starting out in both vertices of the probability simplex: $T'_0 \in \{0,1\}$.*

Proof. There are two boundary points Z_1, Z_2 of the region C (see Figure 3.2.1). Let $Z_1 \leq T_0 \leq Z_2$. By the same argument as in Theorem C.1.2, all the iterates T'_i lie between the starting point T'_0 and T_0. Therefore, the algorithm has to converge to Z_1 from the starting point $T'_0 = 0 \leq Z_1$ and to Z_2 from the starting point $T'_0 = 1 \geq Z_2$. □

Moreover, it is possible to establish complexity bounds for this type of MDPs since the following known result establishes that one-dimensional Newton's method makes progress in each iteration by either significantly decreasing the height or increasing the slope of the function $U^{\Pi_T}(T)$:

Lemma C.1.4. *Let $U^{\pi}(T) = m^{\pi}T + r^{\pi}$ be a linear function of a single variable T for all policies π. Let $U^{\Pi_{T_0}}(T) = m^*T + r^*$ denote the value function of the optimal policy at T_0 and $U^{\Pi_{T'_i}}(T) = m_iT + r_i$ denote the value function of the optimal policy in the iteration i of Newton's method. Let $\delta_i = U^{\Pi_{T'_i}}(T'_i) - (m^*T'_i + r^*)$ be the height of the function $U^{\Pi_{T_i}}(T_i)$ with respect to $U^{\Pi_{T_0}}(T'_i)$ at iteration i and m_i be its slope. Then either $\frac{m^*-m_{i+1}}{m^*-m_i} \leq \frac{1}{2}$ or $\frac{\delta_{i+1}}{\delta_i} \leq \frac{1}{2}$.*

The proof of a slightly modified version of this lemma appears in [52]. Before we proceed, we need the following definition which captures how closely one policy can resemble another as the transition probabilities T vary.

Definition C.1.5. The distance between two policies π and π' is defined as $max_T|U^{\pi}(T) - U^{\pi'}(T)|$.

We can now state a global bound on the complexity of one-dimensional Newton's method:

Theorem C.1.6. *Let γ be the smallest distance larger than 0 between any policy and Π_{T_0}. Let the MDP have bounded rewards: $|R_0(s,a)| \leq M$ for $\forall s, a$. Then, after $t = O(\log(\frac{M}{\gamma\epsilon(1-\beta)}))$ iterations, the iterates $T'_{i \geq t}$ of Newton's method are within ϵ of the limit point T'_{∞}.*

Proof. Since rewards are bounded by M, discounted rewards at any state for any $T \in [0,1]$ are bounded by $\frac{M}{(1-\alpha)}$. Therefore, $m^* - m_i \leq \frac{4M}{(1-\alpha)}$. Distance between two linear policies $U^{\Pi_{T_i}}(T) = m_iT + r_i$ and $U^{\Pi_{T_0}}(T) = m^*T + r^*$ on $T \in [0,1]$ is equal to $max_{T\in[0,1]}(|m_iT + r_i - (m^*T + r^*)|) = max_{T\in\{0,1\}}(|m_iT + r_i - (m^*T + r^*)|) = max(r_i - r^*, m^* + r^* - (m_i + r_i)) \leq m^* - m_i$ because $U^{\Pi_{T_i}}(T)$ and $U^{\Pi_{T_0}}(T)$ intersect

at $T'_\infty \in [0,1]$ (and applying $max(a,b) \leq a+b$ for nonnegative a,b). Therefore, $m^* - m_i \geq \gamma$. Combining the lower and upper bounds on $m^* - m_i$ and its geometric convergence from Lemma C.1.4, we get that the value of m_i changes in at most $\log(\frac{4M}{\gamma(1-\beta)})$ iterations. Hence, δ_i must change in the other iterations. It follows from bounded rewards that $\delta_i \leq \frac{2M}{(1-\beta)}$, which, together with the geometric convergence of δ_i to zero from Lemma C.1.4, implies that $\delta_i \leq \gamma\epsilon$ after at most $\log(\frac{2M}{\gamma\epsilon(1-\beta)})$ additional iterations. Since $U^{\Pi_{T_i}}(T)$ and $U^{\Pi_{T_0}}(T)$ intersect at T'_∞ , $\frac{\delta_i}{T'_\infty - T'_i} = m^* - m_i$ which, combined with $m^* - m_i \geq \gamma$ and $\delta_i \leq \gamma\epsilon$ yields $T'_\infty - T'_i \leq \epsilon$. □

C.2 Convergence of Optimal Active Reinforcement Learning

We will prove the following theorem:

Theorem C.2.1. *Let T define the probability simplex of any action a of an MDP with the discount factor α and rewards in the interval $[-M, M]$. Let P be a point set on the probability simplex T with dispersion $d(P) < \epsilon$, and let $B(T_0(\cdot|s,a), R)$ be a sphere centered at $T_0(\cdot|s,a)$ determined by the algorithm above (with R given by the return value in step (3)). Then, for any point $Z \in B(T_0(\cdot|s,a), R)$, the policy $\Pi_{T_0(\cdot|s,a)}$ is not too suboptimal in the world with transition probabilities of action a perturbed to Z. More formally, its utility is not much worse than the utility of any other policy π: $U^{\Pi_{T_0(\cdot|s,a)}}(Z) > U^{\pi}(Z) - 2K\epsilon$, where K is given by $\frac{2\alpha M}{(1-\alpha)^2}$.*

First, we need the following lemma:

Lemma C.2.2. *Let P be a point set with dispersion $d(P) < \epsilon$ on some convex set B. Suppose that $f(u) > 0$ for any $u \in P$ and $\frac{|f(x)-f(y)|}{\|x-y\|} < \phi$ (Lipschitz condition) for any pair of points $x, y \in B$. Then, for any point $z \in B$, $f(z) > -\phi\epsilon$.*

Proof. Take any $z \in B$. Then, by the dispersion condition, there is a $u \in P$ such that $\|u - z\| < \epsilon$, hence, by the Lipschitz condition, $|f(u) - f(z)| < \phi\epsilon$. $f(u) > 0$ then implies that $f(z) > -\phi\epsilon$. □

We will now prove that the Lipschitz condition holds for MDPs, with the Lipschitz constant ϕ bounded by a function of the maximum reward M and the discount factor α:

Lemma C.2.3. *Let T define the probability simplex of any action a of an MDP with the discount factor α and rewards in the interval $[-M, M]$. Define $f_{\pi_1,\pi_2}(X \in T) \triangleq U^{\pi_1}(X) - U^{\pi_2}(X)$. for any two policies π_1, π_2. Then, for any two sets of transition probabilities $T_1, T_2 \in T$, $\frac{|f(T_2)-f(T_1)|}{\|T_2-T_1\|} < \frac{2\alpha M}{(1-\alpha)^2}$.*

Proof. The gradient of the value of any policy $|\nabla_L V^\pi(Z)| < \frac{\alpha M}{(1-\alpha)^2}$ for any Z on the line connecting T_1 and T_2 and direction $L = \frac{T_2 - T_1}{\|T_2 - T_1\|}$. This can be seen as follows: using Lemma C.1.1, we can rewrite the gradient as $\nabla_L V = \alpha P_0^\pi \nabla_L V + \alpha L V_0$ (assuming $\|L\| = 1$). This is the Bellman backup equation for an MDP with probability transitions given by P_0^π and rewards given by LV_0. Since the rewards are bounded by M, the value function αV_0 is bounded by $\frac{\alpha M}{(1-\alpha)}$. Discounting by α again to get a bound on $\nabla_L V(Z)$ yields $\frac{\alpha M}{(1-\alpha)^2}$. The triangle inequality implies that $|\nabla_L f_{\pi_1,\pi_2}(Z)| < \frac{2\alpha M}{(1-\alpha)^2}$.

By the Mean Value Theorem, there exists a point N' on the line connecting T_1 and T_2 such that $\frac{|f(T_2) - f(T_1)|}{\|T_2 - T_1\|} = |\nabla_L f_{\pi_1,\pi_2}(N')| < \frac{2\alpha M}{(1-\alpha)^2}$. □

Finally, the following lemma shows that any point set with a small dispersion on some set S also has a small dispersion on the ball $B(N, R) \subseteq S$, where $B(N, R)$ denotes a ball centered at N of radius R.

Lemma C.2.4. *Let P be a point set with dispersion $d(P) < \epsilon < R$ on some set S. Then for any ball $B(N, R) \subseteq S$, the dispersion of the point set inside $B(N, R)$, $d(P \cap B(N, R))$, is less than 2ϵ.*

Proof. Take any point $x \in B(N, R)$. Take a ball $B'(x + \epsilon \frac{N-x}{\|N-x\|}, \epsilon)$. If $\epsilon < R$, then it is straightforward to verify that B' is contained completely inside B. Moreover, there has to be at least one point p' in P inside B' because $d(P)$ is smaller than the radius of B'. Since the center of B' is less than ϵ away from x and p' is less than ϵ away from the center of B, by the triangle inequality $\|x - p'\| < 2\epsilon$. □

Let $B(T_0(\cdot|s,a), R)$ represent the largest ball that fits within the region C in which $\Pi_{T_0(\cdot|s,a)}$ is dominant. Lemma C.2.4 shows that the dispersion of the point set P is small inside B. Lemma C.2.3 shows that the function $f_{\Pi_{T_0(\cdot|s,a)},\pi}(Z) = U^{\Pi_{T_0(\cdot|s,a)}}(Z) - U^\pi(Z)$ (i.e., the difference between the value of the optimal policy at T_0 and any other policy) does not change too rapidly inside B. We know that $f_{\Pi_{T_0(\cdot|s,a)},\pi}(\cdot)$ is nonnegative on P since the policy $\Pi_{T_0(\cdot|s,a)}$ dominates any other policy π inside B by definition. Plugging all of this into Lemma C.2.2 proves Theorem C.2.1.

References

[1] ATLAS home page. [Online]. http://math-atlas.sourceforge.net/faq.html#NB80.

[2] A subset of the 4 universities dataset containing web pages and hyperlink data used in [12] and available at http://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-51/www/co-training/data/.

[3] P. Abbeel and A. Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *ICML*, 2004.

[4] P. Abbeel and A. Y. Ng. Exploration and apprenticeship learning in reinforcement learning. In *ICML*, 2005.

[5] P. Abbeel, M. Quigley, and A. Ng. Using inaccurate models in reinforcement learning. In *ICML*, 2006.

[6] R. Allan and K. Kennedy. *Optimizing Compilers for Modern Architectures*. Morgan Kaufmann Publishers, 2002.

[7] M. Anthony and N. Biggs. PAC learning and artificial neural networks. Technical report, April 02 2000.

[8] J. Baxter. A model of inductive bias learning. *Journal of Artificial Intelligence Research*, 12:149–198, 2000.

[9] C. Bhattacharyya, K. S. Pannagadatta, and A. Smola. A second order cone programming formulation for classifying missing data. In *NIPS*, 2004.

[10] J. Bilmes, K. Asanović, C. Chin, and J. Demmel. Optimizing Matrix Multiply using PHiPAC: a Portable, High-Performance, ANSI C Coding Methodology. In *Proc. of Int. Conf. on Supercomputing*, Vienna, Austria, July 1997.

[11] C.L. Blake and C.J. Merz. 20 newsgroups database available at http://people.csail.mit.edu/people/jrennie/20newsgroups/, 1998.

[12] Blum and Mitchell. Combining labeled and unlabeled data with co-training. In *COLT: Proceedings of the Workshop on Computational Learning Theory, Morgan Kaufmann Publishers*, 1998.

[13] B. Bonet and J. Pearl. Qualitative mdps and pomdps: An order-of-magnitude approximation. In *UAI*, 2002.

[14] P. Boulet, A. Darte, T. Risset, and Y. Robert. (Pen)-ultimate Tiling? In *INTEGRATION, the VLSI Journal*, volume 17, pages 33–51. 1994.

[15] Ronen I. Brafman and Moshe Tennenholtz. R-MAX - A general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3:213–231, 2002.

[16] David Callahan, Steve Carr, and Ken Kennedy. Improving Register Allocation for Subscripted Variables. In *Proc. of PLDI*, pages 53–65, 1990.

[17] X. Cao. From perturbation analysis to markov decision processes and reinforcement learning. *Discrete Event Dynamic Systems: Theory and Applications*, 13:9–39, 2003.

[18] B.P. Carlin and T.A. Louis. *Bayes and Empirical Bayes Methods for Data Analysis*. Chapman and Hall, 2000.

[19] S. Carr, C. Ding, and P. Sweany. Improving software pipelining with unroll-and-jam. *Proc. of 29th Hawaii International Conference on System Sciences*, 1996.

[20] C. Chen, J. Chame, and M. Hall. Combining Models and Empirical Search to Optimize for Multiple Levels of the Memory HierarchyA. In *In Proc. of CGO*, pages 111–122, 2005.

[21] S. Coleman and K. S. McKinley. Tile Size Selection Using Cache Organization and Data Layout. In *Proc. of PLDI*. ACM Press, June 1995.

[22] M. Collins. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of 2002 Conference on Empirical Methods in Natural Language Processing*, 2002.

[23] K.D. Cooper and T. Waterman. Investigating Adaptive Compilation Using the MIPSPro Compiler. In *Proc. the LACSI Symposium,*, Los Alamos Computer Science Institute, October 2003.

[24] R. Dearden, N. Friedman, and D. Andre. Model based bayesian exploration. In *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence (UAI-99)*, pages 150–159, S.F., Cal., July 30–August 1 1999. Morgan Kaufmann Publishers.

[25] G. DeJong. Explanation-based learning. In A. Tucker, editor, *Computer Science Handbook*, pages 68.1 – 68.18. Chapman & Hall/CRC and ACM, 2nd edition, 2004.

[26] L. Devroye. *Non-Uniform Random Variate Generation*. Springer-Verlag, New York, NY, 1986.

[27] P. Diniz and M. Rinard. Dynamic feedback: An effective technique for adaptive computing. *Proc. of PLDI*, 1997.

[28] R. Duda, P. Hart, and D. Stork. *Pattern Classification*. John Wiley, 2001. 2nd edition.

[29] A. Epshteyn and G. DeJong. Rotational prior knowledge for svms. In *Proceedings of the Sixteenth European Conference on Machine Learning*, 2005.

[30] A. Epshteyn and G. DeJong. Generative prior knowledge for discriminative classification. *Journal of Artificial Intelligence Research*, 27:25–53, 2006.

[31] A. Epshteyn and G. DeJong. Qualitative reinforcement learning. In *International Conference on Machine Learning*, 2006.

[32] A. Epshteyn, M. Garzaran, G. DeJong, D. Padua, G. Ren, X. Li, K. Yotov, and K. Pingali. Analytic models and empirical search: A hybrid approach to code optimization. In *International Workshop on Languages and Compilers for Parallel Computing*, 2005.

[33] Y. Erlich, D. Chazan, S. Petrack, and A. Levy. Lower bound on VC-dimension by local shattering. *Neural Computation*, 9(4), 1997.

[34] T. Evgeniou and M. Pontil. Regularized multi-task learning. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2004.

[35] M. Fink. Object classification from a single example utilizing class relevance metrics. In *Advances in Neural Information Processing Systems*, 2004.

[36] R. Fletcher. *Practical Methods of Optimization*. John Wiley and Sons, West Sussex, England, 1987.

[37] M. Frigo and S. G. Johnson. FFTW: An Adaptive Software Architecture for the FFT. *Proc. IEE Intl. Conf. on Acoustics, Speech, and Signal Processing*, 3:1381–1384, 1998.

[38] G. Fung, O.L. Mangasarian, and J.W. Shavlik. Knowledge-based support vector machine classifiers. In *Advances in Neural Information Processing Systems*, 2002.

[39] R. Givan, S. Leach, and T. Dean. Bounded-parameter markov decision processes. *Artificial Intelligence*, 122(1–2):71–109, 2000.

[40] R. Greiner and W. Zhou. Structural extension to logistic regression: Discriminative parameter learning of belief net classifiers. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence*, 2002.

[41] B. Grunbaum. *Convex Polytopes*. John Wiley, 1967.

[42] A. Gunawardana and W. Byrne. Convergence theorems for generalized alternating minimization procedures. *Journal of Machine Learning Research*, 6:2049–2073, 2005.

[43] P. Hansen, B. Jaumard, and G. Savard. New branch-and-bound rules for linear bilevel programming. *SIAM Journal on Scientific and Statistical Computing*, 13:1194–1217, 1992.

[44] K. Huang, I. King, M. R. Lyu, and L. Chan. The minimum error minimax probability machine. *Journal of Machine Learning Research*, 5:1253–1286, October 2004.

[45] T. Jaakkola, M. Meila, and T. Jebara. Maximum entropy discrimination. In *Advances in Neural Information Processing Systems*, 1999.

[46] T. Jebara. *Machine Learning: Discriminative and Generative*. Kluwer Academic Publishers, 2004.

[47] Thorsten Joachims. Text categorization with support vector machines: learning with many relevant features. In *Proceedings of the Tenth European Conference on Machine Learning*, 1998.

[48] M. Kearns and S. Singh. Near optimal reinforcement learning in polynomial time. *Machine Learning*, 49(2-3):209–232, 2002.

[49] G. R. G. Lanckriet, L. El Ghaoui, C. Bhattacharyya, and M. I. Jordan. Minimax probability machine. In *Advances in Neural Information Processing Systems*, 2001.

[50] A. Laud and G. DeJong. The influence of reward on the speed of reinforcement learning: An analysis of shaping. In *ICML*, 2003.

[51] M. S. Lobo, L. Vandenberghe, S. Boyd, and H. Lebret. Applications of second-order cone programming. *Linear Algebra and its Applications*, 284(1–3):193–228, 1998.

[52] O. Madani. *Complexity results for infinite-horizon Markov decision processes*. PhD thesis, University of Washington, 2000.

[53] O. Madani. On policy iteration as a newton's method and polynomial policy iteration algorithms. In *AAAI/IAAI*, 2002.

[54] O. Mangasarian, J. Shavlik, and E. Wild. Knowledge-based kernel approximation. *Journal of Machine Learning Research*, 5, 2004.

[55] Michael McHale. A comparison of wordnet and roget's taxonomy for measuring semantic similarity. *CoRR*, cmp-lg/9809003, 1998.

[56] G. Miller. WordNet: an online lexical database. *International Journal of Lexicography*, 3(4), 1990.

[57] A. Y. Ng, D. Harada, and S. J. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, 1999.

[58] A. Y. Ng and M. I. Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *Advances in Neural Information Processing Systems*, 2001.

[59] H. Niederreiter. *Random number generation and quasi-Monte Carlo methods.*

[60] M. Puterman. *Markov Decision Processes—Discrete Stochastic Dynamic Programming*, chapter 4.7. John Wiley & Sons, Inc., New York, NY, 1994.

[61] R. Raina, Y. Shen, A. Y. Ng, and A. McCallum. Classification with hybrid generative/discriminative models. In *Advances in Neural Information Processing Systems*, 2003.

[62] C. P. Robert. *The Bayesian Choice.* Springer-Verlag, 1994.

[63] T. Roos, H. Wettig, P. Grunwald, P. Myllymaki, and H. Tirri. On discriminative bayesian network classifiers and logistic regression. *Machine Learning*, 59:267–296, 2005.

[64] R. Sabbadin. A possibilistic model for qualitative sequential decision problems under uncertainty in partially observable environments. In *UAI*, 1999.

[65] B. Scholkopf, P. Simard, V. Vapnik, and A. Smola. Prior knowledge in support vector kernels. In *NIPS.* The MIT Press, 1997.

[66] G. A. F. Seber and C. J. Wild. *Nonlinear Regression.* John Wiley and Sons, Inc., 1989.

[67] M. Shaked and J. G. Shanthikumar. *Stochastic Orders and Their Applications.* Academic Press, San Diego, CA, 1994.

[68] J. F. Sturm. Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. *Optimization Methods and Software*, 11:625–653, 1999.

[69] Q. Sun and G. DeJong. Explanation-augmented svm: an approach to incorporating domain knowledge into svm learning. In *Proceedings of The Twenty Second International Conference on Machine Learning*, 2005.

[70] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, MA, 1998.

[71] J. S. Taylor and P. Bartlett. Generalization performance of support vector machines and other pattern classifiers. In *Advances in kernel methods: support vector learning*, May 1998.

[72] S. Thrun. Is learning the n-th thing any easier than learning the first? In *Advances in Neural Information Processing Systems*, 1995.

[73] M. E. Tipping. Sparse bayesian learning and the relevance vector machine. *Journal of Machine Learning Research*, 1:211–244, 2001.

[74] S. Tong and D. Koller. Restricted bayes optimal classifiers. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence*, 2000.

[75] S. Triantafyllis, M. Vachharajani, N. Vachharajani, and D. August. Compiler optimization-space exploration. *Int. Symp. on CGO*, 2003.

[76] V.N. Vapnik. *The Nature of Statistical Learning Theory.* Springer-Verlag, 1995.

[77] R. Clint Whaley, Antoine Petitet, and Jack J. Dongarra. Automated Empirical Optimization of Software and the ATLAS Project. *Parallel Computing*, 27(1–2):3–35, 2001.

[78] M. Wolfe. Iteration Space Tiling for Memory Hierarchies. In *Third SIAM Conf. on Parallel Processing for Scientific Computing*, December 1987.

[79] X. Wu and R. Srihari. Incorporating prior knowledge with weighted margin support vector machines. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2004.

[80] J. Xiong, J. Johnson, R. Johnson, and D. Padua. SPL: A Language and a Compiler for DSP Algorithms. In *Proc. of PLDI*, pages 298–308, 2001.

[81] K. Yotov and al. A Comparison of Empirical and Model-driven Optimization. *Proc. of PLDI*, pages 63–76, 2003.

[82] K. Yotov, X. Li, G. Ren, M. J. Garzarán, D. Padua, K. Pingali, and P. Stodghill. Is Search Really Necessary to Generate a High Performance Blas? *In Proc. of the IEEE, special issue on Program Generation, Optimization, and Platform Ada ptation*, 23:358–386, February 2005.

[83] Kamen Yotov, Keshav Pingali, and Paul Stodghill. Think Globally, Search Locally. In *ICS '05: Proceedings of the 19th annual international conference on Supercomputing*, pages 141–150, New York, NY, USA, 2005. ACM Press.

[84] W. Zangwill. Convergence conditions for nonlinear programming algorithms. *Management Science*, 16:1–13, 1969.

Vita

Curriculum Vitæ

Arkady Epshteyn

Department of Computer Science, 201 N. Goodwin Ave., 3332 Siebel Center

University of Illinois, Urbana, IL 61801, USA

E-mail: aepshtey@yahoo.com

RESEARCH INTERESTS

Artificial Intelligence, Machine Learning, Statistics, Natural Language Processing, Probabilistic and Discriminative Classification, Probabilistic Planning.

EDUCATION

Ph.D. in Computer Science, University of Illinois at Urbana-Champaign, *expected in* May, 2007

Thesis: *Prior Knowledge for Decreasing Sample Complexity in Classification, Reinforcement Learning, and Compiler Optimization*

Advisor: Gerald DeJong

GPA: 3.90/4.00

M.S. in Computer Science, University of Illinois at Urbana-Champaign, 1998

Thesis: *Learning in Continuous Multidimensional Domains*

Advisor: Gerald DeJong

B.S. (Highest Honors) in Computer Science, University of Illinois at Urbana-Champaign, 1996

GPA: 4.00/4.00

PROFESSIONAL POSITIONS

Research Assistant Dept. of Computer Science, University of Illinois, (2002-present)

- Designed a framework for solving probabilistic planning problems defined in terms of qualitative statements about the world. The framework allowed the domain expert to define Markov Decision Processes (MDPs) in terms of qualitative statements such as "the faster the car is going, the more likely it is to ascend to the top of the hill", interpreted as constraints on transition probabilities. Developed an algorithm (based on policy iteration) to find the optimal policy in qualitative MDPs.

- Explored the use of a linguistic ontology for text classification. Used the ontology to construct an informative prior for classification, and developed a novel processing algorithm based on convex optimization to efficiently integrate this prior into a large-margin classification algorithm. The technique was used to improve small-sample classification accuracy of a state-of-the-art support vector machine classifier.

- Applied Bayesian statistical techniques to compiler optimization. The system was used to optimize linear algebra routines for any hardware platform with minimal empirical testing.

- Trained a robotic system to play a musical saw. The system consisted of two robotic arms controlled through a computer interface.

Teaching Assistant Dept. of Computer Science, University of Illinois, (2001-2002)

- Involved in various aspects of teaching a high-level undergraduate course (CS423, Operating Systems) and a graduate-level course (CS523, Advanced Operating Systems), including designing and grading programming assignments, quizzes, and exams. Graded critical reviews of published papers. Held office hours to address students' questions.

Senior Software Engineer, AI Team Leader Black Pearl (1999-2001)

- Managed a team of engineers responsible for implementing AI techniques.

- Designed and implemented a system to be used by financial institutions for automatically recommending stocks to investors. To enable this technolody, implemented an industrial-strength rule-processing engine in Java that outperformed analogous implementations produced by competitors (including a C++ implementation). It was shown to be both faster (up to an order of magnitude) and more correct than competing products.

- Designed and implemented a probabilistic inferencing algorithm for stock recommendation. The core of this algorithm consisted of a Bayesian network with several propagation algorithms (junction trees, Gibbs sampling).

- Implemented an end-to-end rule processing solution in one week. Implementation involved designing, writing, and debugging a custom compiler for a rule specification language.

AI Software Engineer Stottler Henke Associates, Inc. (1998-1999)

- Developed a decision support system in C++ to be used by the navy aircraft carrier personnel. The system predicted aircrafts' landing trajectories (using a neural network) and correlated them with corrective suggestions (e.g., the aircraft is too low and too far to the right) using fuzzy logic.

PUBLICATIONS

1. A. Epshteyn and G. DeJong. Generative Prior Knowledge for Discriminative Classification. *Journal of Artificial Intelligence Research* vol. 27 (2006), pages 25-53.

2. A. Epshteyn and G. DeJong. Qualitative Reinforcement Learning. *International Conference on Machine Learning*, 2006. Regular paper, Acceptance rate: 20%

3. A. Epshteyn and G. DeJong. Rotational Prior Knowledge for SVMs. *European Conference on Machine Learning*, 2005. Regular paper, Acceptance rate: 11%

4. A. Epshteyn and M. Garzaran and G. DeJong and D. Padua and G. Ren and X. Li and K. Yotov and K. Pingali. Analytic Models and Empirical Search: A Hybrid Approach to Code Optimization. *International Workshop on Languages and Compilers for Parallel Computing.*, 2005.

5. C. Tseng and A. Epshteyn. Correlation of Heterogeneous Data with Fuzzy Logic. *International Conference on Information Fusion*, 1999. Regular Paper.

6. C. Tseng and A. Epshteyn. Modular Learning with Genetic Aggregation (MOLGA) in Data Prediction. *Genetic and Evolutionary Computation Conference* , 1999. Late-Breaking Paper.

7. A. Epshteyn and R. Stottler. Distributed Task Coordination with Truth Maintenance Systems *PC AI*, 1999

PRESENTATIONS

- Paper and poster presentations, "Qualitative Reinforcement Learning," at the Twenty Third International Conference on Machine Learning, 2006.
- Paper presentation, "Rotational Prior Knowledge for SVMs," at the Sixteenth European Conference on Machine Learning, 2005.
- Paper presentation, "Analytic Models and Empirical Search: A Hybrid Approach to Code Optimization," at the Eighteenth International Workshop on Languages and Compilers for Parallel Computing, 2005.
- Paper presentation, "Correlation of Heterogeneous Data with Fuzzy Logic," at the Second International Conference on Information Fusion, 1999.
- Poster presentation, "Modular Learning with Genetic Aggregation (MOLGA) in Data Prediction," at the Genetic and Evolutionary Computation Conference , 1999.

AWARDS

- James Snyder Award (Undergraduate, best sophomore in CS)
- John Pasta Award (Undergraduate, best junior in CS)
- Bronze Tablet (top 3% of graduating class, BS in CS)

SKILLS

- *Programming Languages:* C++, C, Java, Lisp, Scheme, Matlab, Assembler.
- *Operating Systems:* Windows, UNIX.