



MULTI-OBJECTIVE REINFORCEMENT LEARNING
WITH CONCEPT DRIFT

DISSERTATION

Frederick C. Webber, Civilian

AFIT-ENG-DS-17-D-006

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

DISTRIBUTION STATEMENT A: APPROVED FOR PUBLIC RELEASE;
DISTRIBUTION UNLIMITED

The views expressed in this prospectus are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the United States Government.

This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

AFIT-ENG-DS-17-D-006

MULTI-OBJECTIVE REINFORCEMENT LEARNING
WITH CONCEPT DRIFT

DISSERTATION

Presented to the Faculty
Graduate School of Engineering and Management
Air Force Institute of Technology
Air University
Air Education and Training Command
In Partial Fulfillment of the Requirements for the
Degree of Doctor of Philosophy

Frederick C. Webber, B.S.M.E.C.S, M.S.E.E.
Civilian

December, 2017

DISTRIBUTION STATEMENT A: APPROVED FOR PUBLIC RELEASE;
DISTRIBUTION UNLIMITED

AFIT-ENG-DS-17-D-006

MULTI-OBJECTIVE REINFORCEMENT LEARNING
WITH CONCEPT DRIFT

Frederick C. Webber, B.S.M.E.C.S., M.S.E.E.

Committee Membership:

Dr. Gilbert L. Peterson
Chairman

Maj Christina F. Rusnock, PhD
Member

Dr. Laurence D. Merkle
Member

Dr. Bradley C. Boehmke
Dean's Representative

ADEDJI B. BADIRU, PhD
Dean, Graduate School of Engineering and Management

Abstract

Real world environments are non-stationary and include unanticipated changes that contradict what an agent has previously learned. In machine learning, this is called concept drift, a partially-observable change when the environment modifies without notification. Concept drift has two impacts: agents can fail to adapt, or agents may over fit to noise and overwrite previously learned knowledge. These issues are the plasticity-stability dilemma and catastrophic forgetting. Agents in concept drift environments must take steps to mitigate both problems. These problems are compounded because agents typically have multiple tasks to accomplish simultaneously, requiring multi-objective optimization (MOO). MOO requires an agent to learn about more than one feedback signal, and be able to respond to re-prioritization without re-learning.

This work contributes concept drift-aware multi-objective reinforcement learning (CDMORL), a reinforcement learning algorithm that learns in non-stationary multiple objective environments. At each step of CDMORL, the agent predicts the expected utility for each objective given results from an action. The residual between the expectation and the multiple observed rewards is used to compute a change ratio that is compared to the learning rate. A high ratio indicates change, in which case the agent persists the previous concept. It then attempts to recognize a previous concept or, failing that, begins learning a new concept. Testing on a theoretical domain and a realistic stochastic domain shows that the agent consistently outperforms traditional multi-objective reinforcement learning (MORL). It demonstrates that concept drift can be characterized, detected, and recognized using the developed algorithm. It also shows that concept drift detection is more reliable with multiple objectives than with a single objective.

Acknowledgements

This dissertation is dedicated to my loving family and the many people who supported me from both the Air Force Institute of Technology and the Air Force Research Laboratory. Thank you. Thanks also to my savior, Jesus.

Table of Contents

	Page
Abstract	iv
Acknowledgements	v
List of Figures	x
List of Tables	xiii
List of Symbols	xiv
 I. Introduction	 1
1.1 Background to the Research Problem	4
1.2 Hypothesis	6
1.3 Concept Drift-Aware Multi-Objective Reinforcement Learning	6
1.4 Evaluation	8
1.5 Limitations of the Study	8
1.6 Summary	9
 II. Background	 10
2.1 Markov Decision Problems	11
2.2 Reinforcement Learning	11
2.2.1 Q-Learning	12
2.2.2 Dependency of RL on the Markov Assumption	13
2.3 Multi-objective optimization	13
2.4 Multi-objective Reinforcement Learning	16
2.4.1 Exploration and Exploitation	16
2.4.2 Multi-Objective Q-Learning	17
2.5 Concept Drift	18
2.5.1 Types of Drift	19
2.5.2 Types of Learners	20
2.5.3 Instance Selection	21
2.5.4 Ensembles	23
2.5.5 Characterizing Concept Drift	23
2.5.6 Recurrent Concept Drift	25
2.6 Concept Drift and Learning	27
2.6.1 Evolutionary Dynamic Optimization	27
2.6.2 Adaption using Data Structure Management	27

	Page
2.6.3	Dynamic Learning Rates 28
2.6.4	Integrating Concept Drift into RL 29
2.6.5	Adaptive Deep Learning 33
2.7	Summary 34
III.	Concept Drift-aware Multi-Objective Reinforcement Learning . . 35
3.1	Assumptions 36
3.2	Symbols 36
3.3	CDMORL Algorithm Overview 37
3.4	Concept Characterization, Change Detection, and Recognition 40
3.4.1	Batch-based CDMORL 40
3.4.2	Characterizing Samples 41
3.5	Integrating the CDL into MORL 42
3.5.1	Mitigating Catastrophic Forgetting 43
3.5.2	Mitigating the Plasticity-Stability Dilemma . . 43
3.5.3	Decay 44
3.6	Summary 44
IV.	Experimental Evaluation 46
4.1	Measurement 46
4.1.1	Agent Performance Measures 47
4.1.2	Classifier Performance 49
4.2	Algorithm Nomenclature 50
4.3	Experiment Design and Variables 51
4.3.1	Independent Variables 52
4.3.2	Dependent Variables 52
4.4	Environments 52
4.4.1	Stationary MERLIN 53
4.4.2	Non-stationary MERLIN 54
4.4.3	Resource Gathering 55
4.5	Summary 57
V.	Experiment Results 58
5.1	Stationary MERLIN Tests 58
5.1.1	Accuracy Test: CDMORL Agents 58
5.1.2	Accuracy Test: Batch-based Agents 59
5.1.3	Accuracy Test: Single Objective Agents 59
5.1.4	Summary of Accuracy Tests in Stationary MERLIN 62

	Page	
5.1.5	Test for Correct Concept Change Count in Stationary MERLIN	64
5.1.6	Test for Final Concept Count in Stationary MERLIN	65
5.2	Non-Stationary MERLIN Tests	66
5.2.1	Accuracy Test: CDMORL Agents	66
5.2.2	Accuracy Test: Batch-based Agents	71
5.2.3	Accuracy Test: Single Objective Agents	75
5.2.4	Summary of Accuracy Tests in Non-Stationary MERLIN	79
5.2.5	Test for Concept Change Count in Non-stationary MERLIN	80
5.2.6	Test for Final Concept Count in Non-stationary MERLIN	81
5.3	Resource Gathering	82
5.3.1	Accuracy Test: CDMORL Agents	82
5.3.2	Accuracy Test: Batch-based Agents	88
5.3.3	Accuracy Test: Single Objective Agents	93
5.3.4	Summary of Accuracy Tests in Resource Gathering	97
5.3.5	Test for Concept Change Count in Resource Gathering	98
5.3.6	Test for Final Concept Count in Resource Gathering	99
5.4	Overall Analysis of CDMORL	101
5.5	Discussion	102
5.6	Summary	106
VI.	Conclusion	107
6.1	Conclusions	107
6.1.1	Decay	108
6.1.2	Dynamic Learning Rates with WoLF	108
6.1.3	Batches	108
6.1.4	Combinations	108
6.2	Future Work	109
6.3	Impact	110
	Bibliography	111
Appendix A.	CDMORL via Error Signals	117
A.1	Concept Characterization, Change Detection, and Recognition	119
A.2	Integrating the CDL into MORL	121
A.2.1	Mitigating Catastrophic Forgetting	121

	Page
A.2.2 Mitigating the Plasticity-Stability Dilemma . .	121
A.3 Analysis	122
Appendix B. CDMORL via Statistics	125
Appendix C. Comparison of Subsets of MERLIN Results	128
C.1 Comparison of non-Batch Agents	128
C.2 Comparison of Batch Agents	132
C.3 Comparison of Single Objective Agents	136

List of Figures

Figure		Page
1.	Partially-observable Markov decision process	5
2.	Pareto dominance relations in multi-objective optimization . .	15
3.	Overview of CDMORL	38
4.	Stationary MERLIN exp. 4, CDMORL	59
5.	Stationary MERLIN, exp. 4, batches	60
6.	Stationary MERLIN, exp. 4, single objective	61
7.	MERLIN, non-Batch comparison, exp. 7	67
8.	MERLIN, non-Batch comparison, exp. 8	68
9.	MERLIN, non-Batch comparison, exp. 9	68
10.	MERLIN, non-Batch comparison, exp. 10	69
11.	MERLIN, non-Batch comparison, exp. 11	69
12.	MERLIN, non-Batch comparison, exp. 12	70
13.	MERLIN, Batch comparison, exp. 7	71
14.	MERLIN, Batch comparison, exp. 8	72
15.	MERLIN, Batch comparison, exp. 9	72
16.	MERLIN, Batch comparison, exp. 10	73
17.	MERLIN, Batch comparison, exp. 11	73
18.	MERLIN, Batch comparison, exp. 12	74
19.	MERLIN, Single Objective comparison, exp. 7	75
20.	MERLIN, Single Objective comparison, exp. 8	76
21.	MERLIN, Single Objective comparison, exp. 9	76
22.	MERLIN, Single Objective comparison, exp. 10	77
23.	MERLIN, Single Objective comparison, exp. 11	77
24.	MERLIN, Single Objective comparison, exp. 12	78
25.	Resource gathering exp. 13, CDMORL	83

Figure		Page
26.	Resource gathering exp. 14, CDMORL	84
27.	Resource gathering exp. 15, CDMORL	85
28.	Resource gathering exp. 16, CDMORL	86
29.	Resource gathering exp. 17, CDMORL	86
30.	Resource gathering exp. 18, CDMORL	87
31.	Resource gathering exp. 13, Batch-based CDMORL	88
32.	Resource gathering exp. 14, Batch-based CDMORL	89
33.	Resource gathering exp. 15, Batch-based CDMORL	90
34.	Resource gathering exp. 16, Batch-based CDMORL	91
35.	Resource gathering exp. 17, Batch-based CDMORL	92
36.	Resource gathering exp. 18, Batch-based CDMORL	92
37.	Resource gathering exp. 13, Single Objective CDRL	93
38.	Resource gathering exp. 14, Single Objective CDRL	94
39.	Resource gathering exp. 15, Single Objective CDRL	94
40.	Resource gathering exp. 16, Single Objective CDRL	95
41.	Resource gathering exp. 17, Single Objective CDRL	95
42.	Resource gathering exp. 18, Single Objective CDRL	96
43.	Stationary MERLIN, exp. 1, CDMORL	128
44.	Stationary MERLIN, exp. 2, CDMORL	129
45.	Stationary MERLIN, exp. 3, CDMORL	129
46.	Stationary MERLIN, exp. 4, CDMORL	130
47.	Stationary MERLIN, exp. 5, CDMORL	130
48.	Stationary MERLIN, exp. 6, CDMORL	131
49.	MERLIN, Batch comparison, exp. 1	132
50.	MERLIN, Batch comparison, exp. 2	133
51.	MERLIN, Batch comparison, exp. 3	133
52.	MERLIN, Batch comparison, exp. 4	134
53.	MERLIN, Batch comparison, exp. 5	134

Figure		Page
54.	MERLIN, Batch comparison, exp. 6	135
55.	MERLIN, Single Objective comparison, exp. 1	136
56.	MERLIN, Single Objective comparison, exp. 2	137
57.	MERLIN, Single Objective comparison, exp. 3	137
58.	MERLIN, Single Objective comparison, exp. 4	138
59.	MERLIN, Single Objective comparison, exp. 5	138
60.	MERLIN, Single Objective comparison, exp. 6	139

List of Tables

Table		Page
1.	Comparison of MDP, POMDP, RL, and CD-RL	11
2.	CDMORL Parameters	37
3.	Summary of algorithms	51
4.	CDMORL Parameters, MERLIN-Specific	54
5.	CDMORL Parameters, Resource Gathering-Specific	56
6.	Accuracy for stationary MERLIN experiments	63
7.	False Positives in stationary MERLIN	64
8.	Statistical Significance of concept change count in stationary MERLIN	65
9.	Concept count in stationary MERLIN	66
10.	Statistical Significance of final concept count in stationary MERLIN	66
11.	Accuracy for non-stationary MERLIN experiments	79
12.	False Positives in non-stationary MERLIN	80
13.	Statistical Significance of false positives in non-stationary MERLIN	80
14.	Concept count in non-stationary MERLIN	81
15.	Statistical Significance of final concept count in non-stationary MERLIN	82
16.	Accuracy for Resource Gathering experiments	97
17.	False Positives in Resource Gathering	98
18.	Statistical Significance of false positives in resource gathering	99
19.	Concept count in Resource Gathering	99
20.	Statistical Significance of final concept count in resource gathering	100
21.	CDMORL Error Signal Version Parameters	118
22.	CDMORL Statistical Version, Parameters	126

List of Symbols

Symbol		Page
$\mathcal{S}$	set of states	11
$\mathcal{A}$	set of actions	11
$R(s, a)$	reward as a function of state and action	11
$T(s, a, s')$	transition probability	11
s'	subsequent state	11
a	action	11
s	state	11
r	instantaneous reward	11
p	probability	12
$Q(s, a)$	utility as a function of state and action	12
q	instantaneous utility value	12
$\pi(s, a)$	probability of taking an action in a state	12
γ	discount on future reward	12
α	learn rate	12
ϵ	exploration rate	13
x	solution, often of real numbers	14
$\mathbf{X}$	set of solutions	14
f	single objective function	14
$\mathbf{F}$	objective function set	14
c	concept (class)	19

MULTI-OBJECTIVE REINFORCEMENT LEARNING WITH CONCEPT DRIFT

I. Introduction

Simple, every day problems solved trivially by humans are difficult or time-consuming to explain well enough for a robot to accomplish the task sufficiently. An example is a robot cleaning a floor. For a robot to clean a floor, one approach is to provide detailed floor plans and daily schedules of when the room would be in use. This approach means every robot needs custom instructions for each location that are updated for each day, making it far more work to use a robot than for a human to simply perform the task on their own. A much simpler approach is to control the robot with an intelligent agent capable of generalized learning. An intelligent agent reasons about its action choices to optimize its performance in an environment. For example, an intelligent agent controlling a floor cleaner is able to balance the tasks of picking up as much dirt as possible, minimizing energy usage (by only picking up dirt when the floor is sufficiently dirty), and ensuring the robot doesn't run out of energy, all based on a few basic sensors such as dirt accumulated, battery life, and communications with a charging station. Modern consumer-grade floor cleaning robots explore the environment randomly, making it easy to move from room to room or building to building and still satisfy the owner. However, robots designed for much more complicated tasks such as driving on roads require adaptation.

Say that an intelligent agent controlling a car learns on sunny, dry roads in the middle of a desert. What happens when it is dark and wet? How about when it is in a tunnel or on wet leaves? Then what happens when the agent returns to sunny, dry roads? Traditional designs for intelligent agents cannot handle these sorts of problems effectively because there is no way to characterize environmental change and recognize environments.

Other fields have established tools that characterize environmental change and recognize environments, such as those backing product recommenders in online shopping [75]. These tools attempt to classify the type of shopper to provide product recommendations likely to be of interest. If the user searches for a book about a young woman running an orphanage, then recommendations will be tailored to a young woman who is possibly a new mother. If the searches about young heroines saving babies gives way to exclusively searching for video games about combat, many recommender systems assume that the individual using the computer is most probably a young man and changes the products recommended again. The recommender system, however, is unable to perform tasks beyond classification (assigning labels), making it inappropriate for driving a robotic car.

What is missing is a union between both the intelligent agent handling many tasks and the recommender system dynamically adapting to changing situations. Such a union would be capable of tasks such as learning to drive a car and then moving to new environments and succeeding.

One type of intelligent agent is a reinforcement learning (RL) agent [61]. A RL agent learns by taking actions in an environment and associating observations and actions with feedback signals in the form of a reward (or punishment). For example, an RL agent learning to drive a car has the road as its environment. The agent observes its speed, takes an action in the form of an acceleration (such as depressing the gas pedal), and receives a reward as to how precisely it achieved a specific speed. The agent learns a policy that correlates the expected utility (an estimate of the long term reward) an agent receives when it takes an action given an observation. For example, the car driving agent expects to achieve a high speed when it gives a large acceleration command (such as flooring the gas pedal) when it observes that the car is moving slowly or stopped.

Agents in complex environments such as the real world can have multiple reward signals. For example, an RL agent driving a car has both a fuel efficiency and a

responsiveness to accelerations. Over time, a multi-objective reinforcement learning (MORL) agent learns a policy that optimizes the agent’s expected utility, exploiting to maximize future rewards and exploring to ensure that a better policy doesn’t exist.

When an agent receives rewards that deviate from what has been learned, they experience a phenomenon known as concept drift [66], because the previous concept (relationship between agent and environment) has changed. Concept drift is a partially observable form of non-stationarity. An environment has non-stationarity when cause and effect relationships change among observations, actions, and rewards. A partially observable change is one that the agent cannot directly observe, but one that can be detected when feedback is off nominal. For the car driving agent example, a dry road is a concept, and a wet road is a concept; when a dry road becomes wet, concept drift occurs. The change of a road from dry to wet is partially observable because even though the agent can’t directly observe the change (in this example, the agent only observes the car’s speed), the pattern of cause and effect changes as the car experiences slipping.

Traditional approaches to RL and MORL are predicated on the assumption that the environment is stationary [34], meaning concept drift is excluded. Such agents are ill suited for situations outside of controlled settings [73], because change is inherent to the real world: sensors degrade without notice, actions yield inconsistent results, and the rewards from following a policy changes. RL agents that do not handle non-stationarity will either forget what it learned or not learn at all. RL agents cannot discern an optimal next action in such an environment while only considering their current observation [56]. Traditional RL agents for such non-stationary environments have two challenges: catastrophic forgetting, where the agent inadvertently forgets how to perform in previous circumstances; and the plasticity-stability dilemma, where the learning rate tends to either over correct (too plastic) or under correct (too stable) [26]. An example of catastrophic forgetting is the car agent learning to drive on the dry road, then forgetting how to drive on dry roads after encountering and learning to drive on a wet road. The plasticity-stability dilemma shows up when the

car agent experiences mountainous roads: the agent should adjust its acceleration commands some, as not adjusting at all (too stable) could result in the agent being a traffic hazard and over-tuning (too plastic) will cause problems when the road levels out again.

Concept drift therefore hampers RL agents, requiring a strategy to first assess whether concept drift has happened, and if so, to then mitigate it. This dissertation develops a method of multi-objective reinforcement learning incorporating a concept drift learner, analyzes the handling options, and demonstrates their efficacy on a real world environment. Because very little research exists on multi-objective reinforcement learning and virtually no research combining it with concept drift prior to this dissertation, measures are adapted from related fields of research and new domains are defined for testing.

The research described here mitigates both catastrophic forgetting and the plasticity-stability dilemma in multi-objective non-stationary environments by ascertaining whether integration of a concept drift learner (defined in Section 2.5) into multi-objective reinforcement learning can provide the information necessary to handle the changes most effectively.

1.1 Background to the Research Problem

Reinforcement learning (RL) is a set of algorithms that focus on agents learning an optimal action policy while interacting with their environment to earn rewards [65]. Reinforcement learning agents represent their environment as Markov decision process (MDP), which consist of a state space, action space, transition probability from one state to another given an action, and a reward mechanism. Agent observations are states, in which their actions yield rewards and possibly a different state. MDPs are predicated on the assumption that the current state completely describes the environment [61]. An extended form of an MDP is the partially-observable MDP (POMDP), which adds a belief state [46] because the full state cannot be observed. Figure 1 depicts a POMDP. MDPs with concept drift (with discrete concepts) are

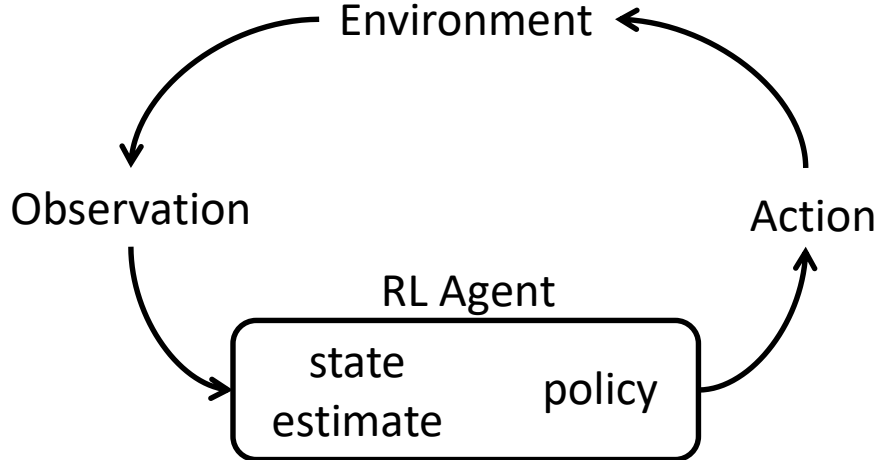


Figure 1: Overview of a partially-observable Markov decision process.

a subset of POMDPs [11], with the belief state effectively designating the current concept. Section 2.2 presents a more in-depth overview of reinforcement learning. For additional information, Sutton and Barto [65], Russell and Norvig [61], and Kaelbling, et al. [34] provide deeper coverage of reinforcement learning.

RL helps agents cope with dynamic environments [18, 31, 49, 54]. Traditional methods of RL assume the task environment is stationary [11]. However, environments in the real world are non-stationary [4]. Some existing RL agent designs address non-stationarity implicitly, often by assuming it doesn’t exist. These techniques handle mild non-stationarity [34], but the result is that these agents must repeatedly relearn a policy as it changes [60]. Such agents must also manage the plasticity-stability dilemma, lest they overreact to noise or fail to adapt. These problem are exacerbated in environments with concept drift.

Concept drift is the phenomenon that occurs when the definition of a classification changes in a manner not directly observable [76]. One example of concept drift in society is the definition of “fuel efficient”: in the past, it may have meant 30 miles per gallon, and in the future it could mean that a car gets better than 60 miles per gallon. In the example of the RL agent driving a car, it means a specific pattern of accelerating, such as “drive like the road is dry.”

Concept drift classifiers examine an input and then apply a label. They can inform agents by performing concept recognition [27] and by concept characterization, which assesses the predictability of the environment. Many existing concept drift classifiers provide explicit characterizations of stable, unstable (or warning), and changed (or drifted) that an RL agent can leverage. A car that typically gets a certain fuel economy is considered to have a stable rate of fuel economy if it gets about the same economy each tank, perhaps give or take 10%. The car’s fuel efficiency is questionable (unstable) if, for no apparent reason, it decreases by 20%. The car’s fuel efficiency has unquestionably degraded if it is suddenly getting 50% of the fuel economy that it had been - its fuel efficiency has drifted (and the car merits further inspection).

1.2 *Hypothesis*

The hypothesis in this dissertation is that by using concept drift learning in MORL, the agent achieves lower (better) rewards. This hypothesis is predicated on the idea that an agent with concept recognition can mitigate catastrophic forgetting by restoring previous information, and on the idea that an agent with information on concept characterization can mitigate the plasticity-stability dilemma by reacting to changes in the environment.

1.3 *Concept Drift-Aware Multi-Objective Reinforcement Learning*

The algorithm developed is concept drift-aware multi-objective reinforcement learning (CDMORL). CDMORL is a novel algorithm that explicitly addresses environments with non-stationarity through incorporating of concept drift learning (CDL) into multi-objective reinforcement learning (MORL). CDMORL includes three methods of concept characterization (one of which is the main focus, while the other two are briefly analyzed in the appendices to identify lessons learned).

This dissertation contributes:

1. the use of convergence measures to explicitly characterize concept stability, detect concept drift (changes in concept), and recognize concepts in environments with one or more objectives;
2. a way to mitigate catastrophic forgetting by saving a learned policy when concept drift is detected and restoring that learned policy when that concept is recognized again; and
3. three strategies for the plasticity-stability dilemma: decaying learning rates, adjusting learning rates, and waiting to process observations, actions, and rewards in small batches.

CDMORL maintains a set of concept models. Each concept model consists of a concept drift learner, a history of how well that concept model matches the environment, and the agent policy corresponding to that concept. One concept model is active at a time, meaning only that concept drift learner and that policy learn, while the stored concept drift models do not have any components that learn. Each step, the environment concept stability is characterized by the concept drift learner (the classification approach is a topic covered in Section 3.3). When the characterization is “drift,” the current concept model is set inactive. If any other models indicate a match, that model is said to have recurred and it is set as the active model, otherwise a new model is created. Methods the agent can to mitigate concept drift by using the concept characterization and recognition are described in Chapter III, which include changing the learning rate when the characterization is unstable; decaying the learning rate of an agent¹; and processing inputs of observation, action, and reward in small batches to improve accuracy.

¹changing and decaying are described in Chapter III, with changing meaning an increase or decrease and decaying meaning a permanent reduction in amplitude; the two are not mutually exclusive as formulated here

1.4 *Evaluation*

The algorithm is experimentally validated in three different environments. The first environment is a stationary multi-objective problem designed to demonstrate whether integrating a concept drift classifier degrades MORL. The second environment is a non-stationary multi-objective problem, designed to require use of CDMORL to achieve the best results. The third environment is a resource gathering task, which is a non-stationary multi-objective problem more representative of a real world task. Because the study of concept drift in RL has only a few papers and the study of concept drift in MORL is entirely novel, this dissertation contributes methods for measuring performance of the MORL component and the performance of the concept drift learner. The four evaluations include:

1. Is CDMORL more effective than MORL? (The measure of effectiveness is defined in Section 4.1 using a specialized formulation of *accuracy*) CDMORL is more effective than MORL (with or without decay).
2. Which concept drift handling techniques are most effective? (Using the same measure of effectiveness) CDMORL with decay and no batch processing is the most effective.
3. Which multi-objective concept drift handling technique most accurately identifies the correct number of concept changes? CDMORL with the combination of decay and batch processing most accurately identifies the correct number of concept changes.
4. Which multi-objective concept drift handling technique most accurately identifies the correct number of concepts? CDMORL with the combination of decay and batch processing most accurately identifies the correct number of concepts.

1.5 *Limitations of the Study*

A limitation of CDMORL is that there is no mechanism for forgetting a learned model, which is desirable for example if duplicates are created due to a failure to

recognize a concept. CDMORL uses a discrete state space, and while converting to an alternative representation such as fuzzy sets or k-nearest neighbors is straight forward, some components, such as the drift characterization and change detection would need to be further developed.

Limitations of the experiment include that several key comparative algorithms were not entirely reproducible: the methods as described were enough to get close, but not enough to say with surety that they were reproduced. A further limitation is that the results are empirical and based on a necessarily finite subset of the infinite set of existing problems to which the algorithm can be applied.

1.6 Summary

This introduction described the need for concept drift-aware multi-objective reinforcement learning. Traditional reinforcement learning and multi-objective reinforcement learning agents are designed for a single concept - a stationary environment. In complex tasks with non-stationarity, concept drift learning can inform an agent so that it can mitigate catastrophic forgetting and the plasticity-stability dilemma.

The remainder of this dissertation presents the background needed to understand the problem, an algorithm to solve the problem, and a thorough analysis of the algorithm. The next chapter describes basic concept drift learning, reinforcement learning, and multi-objective optimization. Chapter III presents the proposed algorithm. Next, Chapter IV presents an experiment plan and performance measures. Chapter V presents the results of the planned experiment. The last chapter concludes with recommendations for future work.

II. Background

Autonomous agents operating in the real world need to be resilient to the uncertain dynamics of the environment. Such agents often balance the needs of multiple objectives, which provide additional noise. This chapter reviews the background required to design an agent that is able to both learn multiple objectives while being resilient to uncertain environments.

Reinforcement learning algorithms use the Markov assumption that all an agent needs to know is the present state and that the history of how the agent got there is irrelevant [34]. However, in non-stationary environments, the Markov assumption is invalidated, leading to catastrophic forgetting and learning rate problems owing to the plasticity-stability dilemma.

The field of concept drift provides a way for an agent to explicitly address non-stationarity. Concept drift is a classification learning problem on temporally-ordered data involving a hidden context or concept, such that the correct classification of data changes over time [23, 75]. Concept drift learners (CDLs) characterize the feedback signals from the environment and exploit that characterization to detect changes and recognize recurrent concepts.

Autonomous agents must often complete multiple tasks simultaneously, requiring multi-objective reinforcement learning. This work achieves this goal by extending a basic multi-objective reinforcement learning agent with a concept drift learner.

This chapter details the relevant background, starting with Markov Decision Problems. Next, reinforcement learning, multi-objective optimization, and multi-objective reinforcement learning are presented. Section 2.5 has an overview of Concept Drift, followed by a few selected works that integrate concept drift and learning. The end of the chapter surveys a few alternatives to concept drift and learning, presents a real-world application domain, and concludes with a summary.

2.1 Markov Decision Problems

Markov Decision Problems (MDP) define a class of problems with a set of states, a set of actions, a probabilistic transition function and a probabilistic reward. An MDP relies on the Markov assumption that the transition and reward probabilities depend only on the current state. An Markov Decision Problem is defined with a defined set of states $\mathcal{S}$, a set of actions $\mathcal{A}$, a possibly stochastic reward function $R(s, a)$, and a stochastic state transition function $T(s, a, s')$, where s' is the state resulting from taking action a in s . Together, an MDP is defined by the tuple $\langle \mathcal{S}, \mathcal{A}, R, T \rangle$. Two ways of solving model-based MDPs are value iteration and policy iteration [34].

The partially-observable Markov Decision Problem (POMDP) is an extension of the Markov Decision Problem [33, 46]. One way of solving POMDPs is to use a state estimator [9], which tracks the probability in the form of a belief state b of making an observation from the environment given the last state and action [9]. Learning problems featuring concept drift are the model-free version of POMDP [33, 46]. MDP and POMDP have models, whereas problems requiring learning do not, as summarized in Table 2.1 [11].

Table 1: Breakdown of types of problem, based on whether a model is known and the observability of the environment [11].

Observability	Model Availability	
	Known	Unknown
Complete	MDP	Traditional RL
Partial	POMDP	Hidden-state RL

2.2 Reinforcement Learning

Reinforcement learning (RL) uses the MDP representation to enable an agent to improve its performance with respect to one or more objectives by associating feedback with its observations and actions [34, 65]. An RL agent learns to optimize its behavior for interactions with an environment through trial-and-error [34], gaining information in the form of the tuple $\langle s, a, r, s' \rangle$ where $s, s' \in \mathcal{S}$, $a \in \mathcal{A}$, and $r \in \mathcal{R}$.

Many methods of RL model interaction between actions and states with a state transition function $T(s, a, s')$ and reward function $R(s, a)$. These methods are *model-based* [34] agents that can solve an MDP that either require the transition function or learn the model explicitly. This dissertation focuses on the more general case when T and R are not provided.

RL agents navigate a trade-off between exploration and exploitation [65]. When the agent explores, it searches for the optimal solution. When the agent exploits, it selects its best known solution, perhaps stochastically. The agent is only performing at its best when exploiting, but if a new and better optimum opens up, it will not know unless it explores. One method, known as ϵ -greedy exploration, selects a random action with probability $p < \epsilon$, $\epsilon \in [0, 1]$.

2.2.1 Q-Learning. One model-free RL algorithm is Q -learning, where an agent learns a function $Q(s, a)$, where the q value is the expected utility. The expected utility is a combination of the reward expected from taking a in s , tempered by how well the agent is positioned for rewards later. Agent behavior follows a policy, often a stochastic function of the state, $\pi(s, a)$, which is denoted as $\pi^Q(s, a)$ when tailored to a specific function $Q(s, a)$. A common form of Q -learning uses the Bellman equation [34]:

$$Q(s, a) = Q(s, a) + \alpha \left(r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right) \quad (1)$$

where the learn rate $\alpha \in (0, 1]$ determines how quickly the agent changes and the discount $\gamma \in [0, 1)$ parameterizes the emphasis on future rewards. The learn rate parameter α assists in learning in non-stationary environments by weighting more recent rewards [65]. The Q function converges to optimality Q^* with probability 1 if each action is executed an infinite number of times on an infinite run and α is decayed properly [34]. The policy for Q^* is denoted as π^{Q^*} .

Often, α and ϵ are decayed so that the agent converges [34]. Agents with decay are unsuitable for handling concept drift.

2.2.2 Dependency of RL on the Markov Assumption. A stationary process has a statistical distribution that does not depend on when the process began [43]. A non-stationary process changes over time. Some sources of dynamics in RL that can change include:

- other agents [6], including humans
- objective functions [20, 48]
- domain of variables [20, 48]
- constraints [48]; and
- the environment [4].

Reinforcement learning problems can feature non-stationarity in each of these forms. Some challenges of non-stationarity in reinforcement learning problems include [26]:

1. No baseline to characterize stationarity
2. Partial observability
3. Stochasticity
4. Variance in stochastic processes
5. Non-stationarity at different time scales

Changes that are not directly observable invalidate the agent’s behavior, making it impossible to discern an optimal next action while only considering its current input [56].

2.3 Multi-objective optimization

Agents in the real world can have multiple objectives competing for attention. For example, controlling the water distribution from a reservoir balances flood levels,

drinking water, irrigation, and power generation [52]. This section formally defines basic multi-objective optimization (MOO) concepts.

The goal of MOO is to find one or more solutions $x \in \mathbf{X}$ that optimizes a set of functions $f \in \mathbf{F}$, as summarized in Equation 2 [12]. The convention in this dissertation is to always minimize.

$$\min \mathbf{F}(x) = \min\{f^1(x), f^2(x), \dots, f^m(x)\}. \quad (2)$$

In non-trivial MOO problems, achieving more optimal performance in one objective happens at the expense of another [68]. Thus, no single strategy can reduce the information to a single objective and be applicable in all circumstances. Because no one solution performs best at all objectives, solutions are compared in terms of dominance relations. For a problem with m objectives:

- A solution x^a **dominates** solution x^b if, $\forall i \in [1, m], f^i(x^a) \leq f^i(x^b)$ and $\exists j f^j(x^a) < f^j(x^b)$. Here, a and b designate different solutions $a \neq b$, and i and j identify different objective functions.
- Solutions are **incomparable** when they are different but neither dominates the other, that is, $\exists i \exists j i \neq j | f^i(x^a) < f^i(x^b), f^j(x^a) > f^j(x^b)$.
- The **Pareto front** consists of all solutions that are not dominated by another solution.
- Solutions that are not dominated and on the Pareto Front are **non-dominated**

The *true Pareto front* is the set of solutions where it is impossible to improve in one objective without sacrificing performance in another [12]. Each objective has a globally optimal value. The vector of these optimal values, even if the solution is not actually achievable, is called the utopia point. These relations are illustrated in Figure 2.

One challenge in MOO is characterizing the quality of solutions in a concise manner. Since scalar values are easy to compare, a number of *indicators* have been

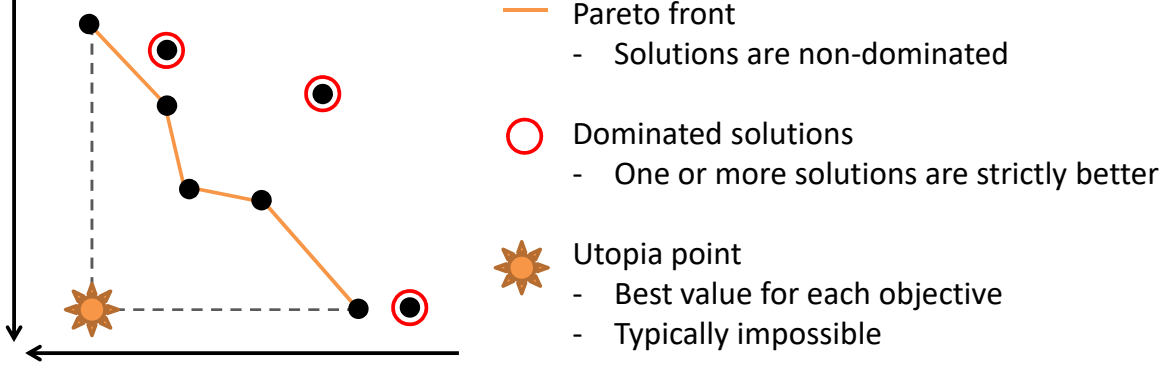


Figure 2: Dominance relations for a set of solutions.

developed to characterize the quality of a solution concisely [12]. Indicators scalarize multiple objective values in a way to quickly compare solutions, often with respect to a reference point $\mathbf{p}_{ref}$. Two such indicators are R2 and Hypervolume. (This work focuses on the special case of only having a single solution, and so formulas have been simplified.)

The R2 indicator functions on an approximation or subset of the known Pareto front [7, 28]. For a single solution, the R2 indicator is:

$$R2(\Lambda, z^*) = \frac{1}{|\Lambda|} \sum_{\lambda \in \Lambda} \max_j \lambda_j |z_j^* - b_j| \quad (3)$$

where b is the vector of values to evaluate, Λ is a set of vector weights corresponding to the objectives, z^* is a solution to compare against. For a single weight vector, it becomes $R2 = \vec{\lambda} \cdot (\vec{r} - z^*)$.

The hypervolume indicator can be applied to a set of solutions or to a single solution. For a set of solutions, it is the volume of the full set, ensuring no space is double counted. The hypervolume is with respect to a reference point, typically an unreachable point that is a little better than the utopia point. The Hypervolume indicator is the product of the objective values:

$$HV = ||\vec{r} - z^*|| \quad (4)$$

Indicators that preserve dominance order relations are Pareto compliant [12]. That is, if one solution set strictly outperforms another, the superior solution set will always have a lower (better) value than the worse set. R2 is not Pareto compliant, but Hypervolume is. For other indicators, see [12].

2.4 *Multi-objective Reinforcement Learning*

Multi-objective reinforcement (MORL) is an online learning paradigm of MOO. Learning extends normal RL by incorporating data from more than one measure. As in MOO, non-trivial MORL problems feature a trade off among objectives, resulting in a set or space of solutions [58,68]. Thus, no single strategy can reduce the information to a single objective and be applicable in all circumstances. For MORL, the challenge is deciding which action to select when attempting to exploit knowledge.

2.4.1 Exploration and Exploitation. Multi-Objective Reinforcement Learning (MORL) differs from RL chiefly in that it is based on multi-objective MDPs (MOMDP) where there are multiple objectives competing for attention, causing the agent to receive $\vec{r} \in \vec{R}$ and learning $\vec{q} \in \vec{Q}$. This vector has n rewards, one per objective, rather than a scalar [53]. Exploration in MORL is able to use naïve ϵ -greedy exploration strategy just like in single-objective RL because this strategy is independent of the objective functions. However, the exploitation strategy is directly impacted by having more than one objective. When exploiting, these values must be combined with an arbitrary scalarization function to produce a single action decision. Previous methods include minimizing the gradient of the utility [51], optimizing one objective while maintaining a performance threshold for all other objectives [47], and maintaining a ratio between objective values [14].

This action selection method is a linear scalarization function, which can only find solutions on the convex hull [70]. The Chebyshev function, which is nonlinear, is able to overcome the flaw of linear functions in that it can find Pareto optimal

solutions with non-convex solution spaces [70]:

$$Chebyshev(x) = \min_i w_i \|f_i(x) - f_i^{best} - \epsilon\| \quad (5)$$

where ϵ is a small number (different from elsewhere in this dissertation), f_i^{best} is the globally best observed value for a given objective, and w_i is a weight for the i^{th} objective.

2.4.2 Multi-Objective Q-Learning. The baseline multi-objective reinforcement learning algorithm for this research is multi-objective Q -learning (MOQL) as done by Van Moffaert, et al. [69] (Algorithm 1). Expected optimal actions are identified by computing an indicator of the expected q values $\vec{q}$ for the current state and proposed action.

MOQL makes action decisions based on a scalarization function, of which it tests a weighted sum, the Chebyshev function (see Equation 5), and the hypervolume indicator [69] (see Equation 4). In terms of agent performance, Chebyshev outperforms hypervolume early on, and the hypervolume tends to overtake Chebyshev after enough runs.

Algorithm 1 Multi-objective Q-Learning [69]

```

1: Initialize  $Q(s, a, o)$  arbitrarily
2: Initialize state  $s$ 
3: repeat
4:   Choose  $a$  using a policy with suitable exploration
5:   Take  $a$ , observe  $s' \in S$ ,  $\vec{r} \in \vec{R}$ 
6:    $\vec{q}_{s,a} \leftarrow \mathbf{Q}(s, a)$ 
7:   Compute expected optimal  $a'$  using scalarization function on  $\vec{q}$ 
8:   for each objective  $o$  do
9:      $Q(s, a, o) \leftarrow Q(s, a, o) + \alpha (\vec{r}_o + \gamma Q(s', a'_{max}, o) - Q(s, a, o))$ 
10:  end for
11:   $s \leftarrow s'$ 
12: until terminal

```

Chatterjee, et al. [10] examine properties of multi-objective MDPs (MOMDPs), presenting several theorems on Pareto optimality and on multi-objective linear pro-

gramming for MOMDPs. Another study on multi-objective MDPs demonstrates that the optimal behavior for a multi-objective problem that will have an action selected based on a linear combination of objective values is the convex hull [3]. For a resource-gathering problem varies based on objective weights, they find a convex hull for six distinct Pareto optimal behaviors, allowing it to rapidly find the optimal solution for a given weight vector. This approach is constrained to problems with a known model and agents selecting actions based on a linear combination of objectives and, while it can handle non-stationarity in action selection, that is the only type of non-stationarity it handles. A survey on empirical evaluation for MORL identified three simple test worlds: MO-Puddleworld¹, MO-MountainCar², and Deep Sea Treasure³ [68].

2.5 *Concept Drift*

Concept drift [23, 66, 73, 75] is a data classification learning problem on non-stationary temporally-ordered data involving a hidden context (hereafter, a concept), such that the correct classification of data changes over time. A concept drift learner (CDL) merely classifies its environment, it does not act on it. A CDL can characterize, detect change, and recognize patterns known as concepts on the latest input in streams of data.

Some examples of concepts includes weather prediction [55]; a predator identifying a rabbit that changes color each season [55]; consumer interests [23]; degrading sensor or effector performance [23]; and undocumented changes in measurement technique [50]. While concept drift is typically temporal, one example of non-temporal concept drift is training a tree classifier when the type of forest is unknown [29].

¹In multi-objective Puddleworld, the agent moves around a grid, don't fall in water, find treasure in fewest steps

²In multi-objective Mountain Car, the car agent must build up potential energy by backing up, then drive forward to get out of a valley (requiring help from the potential energy build up). It must minimize reverse commands, minimize forward commands, and minimize overall time.

³An agent must navigate a grid and minimize time and maximize treasure found. Treasure is spread out in such a way that the lowest values are closest to the start and the greatest values are closest to the end.

This section gives an overview of types of drift followed by types of learners. Using those definitions, the section gives a brief survey of two main types of concept drift learners, instance based methods and ensemble methods. After that, the section explains how concepts characterize and detect change, followed by an overview of how concept recognition is performed.

2.5.1 Types of Drift. Concept Drift occurs when the classification of a specific input changes: that is, $p_t(y|x)$ differs from $p_{t+1}(y|x)$. In Bayesian terms [75],

$$p(c_i|\mathbf{X}) = \frac{p(c_i)p(\mathbf{X}|c_i)}{p(\mathbf{X})}, \quad (6)$$

where c_i is the i^{th} class c and X is the current sample. In these terms, concept drift may occur in three ways [75]:

1. The priors of each class $p(c)$ may change over time. Using traffic as an example, the probability of the classification of *crowded* roads increases as a village grows into a city.
2. One or more classes may have a change in distribution $p(\mathbf{X}|c)$, or virtual drift [23]. To someone from a municipality with a greatly different traffic flow, *crowded* may mean many more or fewer vehicles on the road.
3. Change in the posterior distributions of the class memberships $p(c|\mathbf{X})$. This is called real concept drift. When the number of vehicles on the road is substantially greater or lesser than it once was, and is considered *crowded* by locals, concept drift has occurred.

By distinguishing between real drift and virtual drift, a concept drift learner is more robust to noise [23, 41].

Drifts have different characteristic patterns [23]. Learner algorithms often specialize on a specific type of drift. The types of drift examined in this work are *sudden*, *recurring*, and *outlier*. *Sudden* or *abrupt* drift, also called concept shift, happens without warning. For example, car accidents interrupt traffic immediately. Drifts that are

recurring happen more than once, such as seasons. *Outlier* drift is not concept drift, but is noise or a fault. Drift types not examined in this work include *incremental* and *gradual* drift. *Incremental* drift occurs when one concept blends or fades into another. An example of this is the ramp up and draw down of a traffic surge. A *gradual* drift switches between concepts. As construction delays and closures come and go, they are an example of gradual drift.

2.5.2 Types of Learners. In this research, concepts are represented as a list of continuous numerical variables, such as in [42, 54], which can be described with a mean and variance.

To scope this literature review, the following distinctions are made:

- **Evolutionary vs Adaptive CDLs:** Evolutionary concept drift classifiers are regression models, not seeking to explain or extract other information, just to label a sample as “part of the current class” or not. An example of an evolutionary classifier is what people consider fashionable. This work is only concerned with adaptive classifiers, which attempt to explicitly detect change, separating into *stable* and *drifted* labels. Some adaptive classifiers also attempt to classify a *warning* label when the environment seems to be on the edge of drift.
- **Online vs offline CDLs:** many CDLs function online (while an agent is active), but some methods are designed for offline (post-processing) use only.
- **Classifiers capable of recognition vs those incapable:** adaptive classifiers maintain distinct definitions of classes. Some retain previous concept definitions and are capable of identifying when they recur.
- **Unsupervised vs Semi Supervised / Supervised:** many concept drift classifiers depend on receiving external confirmation of the correct classification (supervised), sometimes for every sample, sometimes for some samples. In re-

inforcement learning, the general case is when there is no teacher, and thus no one to provide the correct classification (unsupervised).

This review emphasizes **online adaptive** CDLs **capable of recognition** using **unsupervised** classification techniques, though other types of classifiers are included if deemed useful, or if they influenced many successors.

2.5.3 Instance Selection. Instance selection methods selectively retain a history of samples, often in the form of a sliding window or a parameter space. The current concept is defined in some manner using the samples in the window or region of the parameter space, though methods exist that can reject individual instances within the window or region.

2.5.3.1 Parameter Space-Based Classifiers. More general instance-based learners operate in feature space, using a representative set of instances to model a concept. A fundamental feature of concept drift learners is that they operate on infinite streams (such as user interactions during online shopping), and can neither retain all points forever, nor can they re-evaluate their entire stream history at each update. To help keep a relevant history, many concept drift learners actively reject individual samples, usually based on a combination of age or relevance to the concept [40]. They can also assign an importance to individual instances [29]. One technique for managing instances includes tracking sample age. This can be used to decay the importance assigned to the instance [29] or to remove the instance [41].

Concept drift algorithms using instance selection emphasize a region of interest in the parameter space to track. Instance selection algorithms add or reject input samples to better describe the set. These learners operate in feature space, using a representative set of instances to model a concept. In some domains, the type and severity of concept drift depends on the region of the parameter space [67]. Some regions may be more stable, allowing different techniques to be used.

2.5.3.2 Windowed algorithms. A windowed learner maintains a recent history of samples, which can help with noise detection, stability characterization, and disambiguation of class labels [54]. The concept is often defined statistically based on a sliding window of samples. This creates a dynamic probability density function. Typically, samples that fit the current concept with high probability are recognized as being members of the same concept.

The multi-resolution learner is a trigger-based learner that uses a set of windows of the most recent inputs, where each window has a different size history (resolution) [41]. The multi-resolution learner characterizes the stability of data in these windows with heuristics that monitor longevity. Samples that are off nominal are considered noise. However, if subsequent samples are consistent with the off nominal sample, the change is instead considered a real drift. The use of multiple windows provides different perspectives, allowing correct detection of different length anomalies.

Several researchers used Bayesian probabilities to detect changepoints [1, 21, 24, 50]. Changepoint detectors identify when in a history a change occurred. A changepoint is an abrupt change in data characteristics. Changepoint detectors retain the last N inputs and makes decisions based on characteristics of the samples, such as mean and variance.

Covariate shift is a problem faced by classifiers that occurs when the training input data follows a different distribution than the operational or test data, but the probability distribution of the expected output of both the training and test is the same distribution. Algorithms that handle this perform covariance adaptation. Some works on covariance adaptation include [45, 62, 63].

2.5.3.3 Batches. Some CDLs employ batch processing [16] to improve accuracy. CDLs using batch processing wait to accumulate multiple samples, which are then analyzed simultaneously to reduce the impact of noise and improve the definition of new concepts. Windowed classifiers are often used in **batch** computations,

where the characterization for multiple samples is computed simultaneously, and the aggregate vote determines the classification of the type of drift (or non-drift). The claim is made in [29] that batch off-line learners are typically ineffective in domains with concept drift due to a homogeneity assumption. The validity of this claim likely depends on the size of the batch and the intensity of the change, as others have had success since then [2, 27].

2.5.4 Ensembles. Ensembles consist of multiple classifiers that vote to determine the current concept [23, 75]. Often, ensembles have a meta learner to perform more advanced voting. Meta learners of ensembles classify in a variety of ways, as demonstrated by the Dynamic Weighted Majority (DWM) method [37, 38]. Methods include majority vote (bagging); weighted majority vote; a weighted majority voting with weighting based on prediction error or expertise (boosting); and using a meta learner that classifies output of child classifiers (stacking) [37]. Meta learners dynamically activate, create or remove ensemble member models, depending on the consistency or relevance of the model to the current data [67]. Some also have ways to dynamically merge or split models. Anecdotally, meta learners generally outperform single classifiers [37].

2.5.5 Characterizing Concept Drift. Concept drift classifiers frequently have three characterizations: stable, warning (or alarm, or drifting), and drifted [22]. These characterizations in turn inform detection and recognition, as well as definition: when a concept is stored, it must be stored with a stable recognition, otherwise the definition will be based on unreliable data. Three examples are provided, all based on statistical measures.

The semi-supervised dynamic fuzzy k-nearest neighbors (SS-DFKNN) [30] was designed to work with only a few samples. It incrementally builds a class definition, maintaining a standard deviation and center of gravity for all the samples in that class. Two indicators show possible drift and actual drift: the percent increase of the standard deviation, and significant movement of the center of gravity. When both

indicators are above a threshold for consecutive steps, the class has changed. When change is decided, the current sample and its k nearest neighbors form the basis of a new class.

A second way of detecting drift is comparing the statistics between samples stored in “recent” and “less recent” sliding windows. The Adaptive Windowing (ADWIN) method [5] defines the size of a total window and two sequential sub-windows, one older and one newer, so that the mean between the two windows differs by no more than a specified threshold. ADWIN increases the size of the windows when the similarity increases and the difference between the two converges, meaning both windows hold the same concept. As the concept diverges, ADWIN shrinks both windows’ sizes. This method has a substantial delay in detecting drift, though it detects drift precisely. The adaptive windowing-based online ensemble (AWOE) [64] manages an ensemble of ADWIN detectors. AWOE declares that drift occurs as soon as any window shrinks, rather than waiting for the older sub-window to be entirely discarded, thus detecting drift much sooner.

The drift detection method (DDM) detects drift based on the statistics of samples within a window [2, 22]. For each input, the algorithms decide based upon a probability p_i that the element is a member of the current concept. The standard deviation decreases or stabilizes as the classifier continues to learn. Thus, a significant increase in the error suggests that concept drift has occurred. Formally, the first algorithm tracks the lowest p and standard deviation s values as p_{min} , s_{min} . Given a large number of samples, the distribution is approximately normal, so confidence intervals from that distribution define two parameters for concept drift warning and change according to the desired confidence interval. Then, with two thresholds for α , if $p_i + s_i \geq p_{min} + \alpha s_{min}$, the concept is said to be unstable or drifted. The early drift detection method (EDDM) computes the distance between two errors by tracking the maximum observed p' and s' . This method compares computes the ratio of $(p'_i + 2s'_i)/(p'_{max} + 2s'_{max})$ and compares that against two thresholds, α and β , which signify *warn* and *drift* characterizations.

2.5.6 Recurrent Concept Drift. Recognizing recurring concepts requires a stable concept definition. Concepts are stored or updated only while stable, to ensure that noise does not contaminate the definition [27, 44, 72]. Accordingly, this means that neither unstable concepts nor destabilizing data points are persisted.

Rule-based classifier FLORA3 [72] checks recent examples against previous concepts when predictive accuracy falters. If a prior concept is more consistent than the present classifier, the prior concept is restored; however, FLORA3 doesn't check previous concepts for a better match until drift is suspected.

When it comes to classification, SS-DFKNN uses k nearest neighbors to compare a sample against the labels of other stored instances [54]. Another method of recognizing previous concepts, for numerical classifiers, is to check for convergence between the centroids of concepts [44].

Meta classifier DWM [38] recognizes previous concepts implicitly based on their classification accuracy within the ensemble. Classifiers phase back in as accuracy increases, just as classifiers failing to remain relevant phase out.

Algorithm 2 provides a unified overview of concept recognition, from the recurrent concept drift (RCD) algorithm [27].

With the ability to characterize, detect change, and recognize concepts, the CDL is able to provide a learning algorithm with the information it needs to defeat catastrophic forgetting and mitigate the plasticity-stability dilemma. The next section provides an overview of previous works in non-stationary learning.

Algorithm 2 Recurring Concept Drift [27] Recognition

```
Initialize max buffer size  $m$ 
Initialize significance parameter  $\alpha$ 
Initialize ensemble size  $e$ 
Initialize classifier list  $C$ 
Create new classifier  $c_a$  and add to  $C$ 
Initialize buffer list  $B$ 
Create new buffer  $b_a$  and add it to  $B$ 
 $c_n \leftarrow null$ 
 $b_n \leftarrow null$ 
repeat
  observe state  $s$ 
   $level \leftarrow detectDrift(c_a, s)$ 
  if  $level == WARNING$  then
    if  $c_n$  is null then
       $Create(c_n)$ 
       $Create(b_n)$ 
    end if
     $SaveFIFO(b_n, s, m)$ 
     $Train(c_n, s)$ 
  else if  $level == DRIFT$  then
    if  $StatTest(C, B, b_n, \alpha)$  then
       $(c_a, b_a) \leftarrow Stored(C, B, b_n)$ 
    else
       $SaveFIFO(C, c_n, e)$ 
       $SaveFIFO(B, b_n, m)$ 
       $c_a \leftarrow c_n$ 
       $b_a \leftarrow b_n$ 
    end if
     $c_n \leftarrow null$ 
     $b_n \leftarrow null$ 
  else
    if  $c_n$  is not null then
       $c_n \leftarrow null$ 
       $b_n \leftarrow null$ 
    end if
     $SaveFIFO(b_a, s, m)$ 
  end if
   $Train(c_a, s)$ 
until terminal
```

2.6 Concept Drift and Learning

RL agents can implicitly handle small amounts of non-stationarity [34]. For changes to the state space, small changes can also be mitigated with discretization, though there is a risk of discretization amplifying the problem. Approximation via fuzzy sets can also alleviate some of the impact of non-stationarity. Other agents use sequential goals [35] as indications of environmental change, but this is directly observable. Beyond these capabilities, several other distinct methods exist, which are reviewed in this section. The first is that of evolutionary dynamic optimization, a subset of evolutionary algorithms. The next part provides extensive but non-exhaustive treatment of reinforcement learning and non-stationary mitigation techniques. The end of the section briefly covers adaptive deep learning.

2.6.1 Evolutionary Dynamic Optimization. Evolutionary dynamic optimization (EDO) is a type of evolutionary optimization that is non-stationary between generations but stationary during a series of evaluations [13, 32, 48]. This dissertation borrows measures from EDO.

EDO optimizes $f(x, t)$, where the space of feasible solutions, the objective functions, and the domain of the objective functions can all vary with time [13]. As of 2011, most research on EDO was based on synthetic test problems, with one popular synthetic domain being moving peaks consisting of a variable domain, range, and maximum [13]. Some EDO works examine multi-objective domains [20, 74]. Results from a prediction-based multi-objective evolutionary algorithm found that, empirically, utilizing historical information to predict solutions that will perform well outperformed random re-initialization [74]. However, typical EDO algorithms are unsuitable for online learning because they learn at the end of an episode or trial rather than along the way.

2.6.2 Adaption using Data Structure Management. Some RL algorithms have built in methods to add and remove knowledge to incorporate information

from new situations. These algorithms dynamically ignore, assimilate and discard information using data structures such as random forests [31] and self-organizing maps [18, 19, 49]. TEXPLORE [31] builds random forests of decision trees using past experiences as leaves, where each leaf is indexed with s and a and has its resulting state s' and experienced reward r . TEXPLORE maintains multiple models through the parallel nature of random forests [31]. TEXPLORE concurrently updates its $Q(s, a)$ and its environment model M while adding $\langle s, a, r, s' \rangle$ tuples to its model. This algorithm relies on a window of the past k actions as inputs to its state prediction model.

Several works use self-organizing maps [18, 19, 49] to incorporate new information. Dynamic Fuzzy Q-Learning [18] learns fuzzy rules as part of a self-organizing map online, matching inputs to a target effector output paired with a Q value. Later expansions of [18] such as the Incremental Topological Preserving Map Fuzzy Q-Learning [19] adjusts fuzzy rule definitions and eliminates rule redundancy.

These algorithms depend on complex mechanisms in data management without explicitly engaging the challenges in non-stationary environments. Another work explicitly attempts to solve the problems affiliated with non-stationarity using a dynamic learn rate, to solve the plasticity-stability dilemma.

2.6.3 Dynamic Learning Rates. One way to address the plasticity-stability dilemma is to use a changing learning rate [6]. This provides stability when performance is as expected and plasticity when performance drops. One such method, the Win or Learn Fast (WoLF) approach [6], extends iterative gradient ascent to learn a policy. WoLF tracks an ‘average’ behavior policy $\bar{\pi}$ in addition to the current policy π . When the expected result from $\bar{\pi}$ outperforms the result from π , the agent is not performing as well as anticipated, and so the learning rate increases to δ_{losing} . Otherwise, the agent is doing well and so uses the more conservative $\delta_{winning}$ rate. WoLF uses policy hill climbing (PHC) [6], later referred to as *WinOrLearnFast-PHC*.

Algorithm 3 Win or Learn Fast

```
1: Initialize  $Q(s, a)$  arbitrarily
2: Initialize state visit counter  $C(s) \leftarrow 0$ 
3:  $\pi(s, a) \leftarrow \frac{1}{|A|}$ 
4: Let  $\delta_l > \delta_w \in (0, 1]$  be learning rates
5: Initialize state  $s \leftarrow s_0$ 
6: repeat
7:   Choose  $a$  according to policy with suitable exploration
8:   Take  $a$ , observe  $s' \in S$ ,  $r \in R$ 
9:   Compute expected optimal  $a'_{max}$ 
10:   $Q(s, a) \leftarrow Q(s, a) + \alpha (r + \gamma Q(s', a'_{max}) - Q(s, a))$ 
11:   $C(s) \leftarrow C(s) + 1$ 
12:  for  $a \in A$  do
13:     $\bar{\pi}(s, a) \leftarrow \bar{\pi}(s, a') + \frac{1}{C(s)} (\pi(s, a') - \bar{\pi}(s, a'))$ 
14:  end for
15:  for  $a \in A$  do
16:     $\delta_{sa} = \min \left( \pi(s, a), \frac{\delta}{|A|-1} \right)$ 
17:    if  $a = a'_{max}$  then
18:       $\Delta_{sa} = -\delta_{sa}$ 
19:    else
20:       $\Delta_{sa} = \sum_{a' \neq a'_{max}} \delta_{sa'}$ 
21:    end if
22:     $\pi(s, a) \leftarrow \pi(s, a) + \Delta_{sa}$ 
23:     $s \leftarrow s'$ 
24:  end for
25: until terminal
```

2.6.4 Integrating Concept Drift into RL. A few have undertaken to integrate concept drift with RL [11, 17, 60]. Their algorithms use multiple models that dynamically activate the most relevant policy. To make concept drift relevant, the environment is assumed to have the following properties [11, 60]:

- the frequency is slow enough that changes can be detected. If there is no regularity, the task becomes impossible [11].
- the concept cannot be directly observed, but it can be estimated based on past state transitions and rewards or other history.
- the agent's actions do not impact the concept.

Additionally, their models assume that there are a finite number of concepts (not necessarily a known quantity) which each have distinct dynamics.

While all previous methods integrating CDLs into RL used ensembles, none were tested in a sufficiently difficult environment to merit use of batches. This work contributes batches to concept drift-aware learning.

2.6.4.1 Concepts as Partially Observable MDP. Choi, et al [11] developed multi-mode reinforcement learning for non-stationary environments with no known model. The algorithm is formulated as a hidden-mode MDP which is transformed into a partially-observable MDP. At each step, the probability of being in mode (concept) m' is estimated from many factors:

- the current mode m ;
- expected number of visits to resulting state s' from previous state s taking action a in mode m ;
- expected number of mode transitions from m to m' ; and
- expected number of visits to mode m .

This approach is novel in the literature in that it considers the likelihood of transitioning from one specific concept to another specific concept. A key component of all three algorithms is utilizing the residuals in the rewards observed by the agents.

2.6.4.2 Estimating Concepts from State Transitions and Rewards.

The problem of multi-modal reinforcement learning (RL) or RL with different concepts involves a task cast in multiple environments that are relatively stationary, but the environment changes arbitrarily without notification [11, 17, 60]. Doya, et al. [17] develop the Multi Model Reinforcement Learning paradigm that develops a responsibility signal to incorporate input from all concepts, which are defined in advance. The responsibility signal accounts for time and the probability of entering a given concept. This signal aggregates all q values in a given state across all concepts. Choi, et al. [11]

formalized a one-way transformation from a hidden mode MDP to a POMDP, along with how to solve it using the Baum-Welch algorithm. Both Choi [11] and Doya [17] learned a probability of transitioning from one concept to another.

Multiple model-based reinforcement learning⁴ [17] uses a meta learner of an ensemble of models. Each model consists of a concept definition and its respective behavior policies. The meta learner weights each model based on the residual between the predicted and actual performance using a credit assignment signal. The credit assignment for each model is predicated on the probability of observing a state transition given the current concept model [17]. Two limitations of this method are that it is model-based and its sub-classifiers do not learn.

2.6.4.3 Explicit Detection of Concept Drift in RL. da Silva [60] presents reinforcement learning with context detection RL-CD, a concept drift-aware reinforcement learning algorithm. The agent assigns an error signal to each concept it maintains. The error signal is a function of the residuals between the predicted and actual transition and reward. At each time step, the concept with the best error signal performance is activated. If the error signal of the best concept drops below a threshold, the concept is considered to have drifted. At that point, a new concept is created. Algorithm 4 summarizes this process.

The concept error E_m is

$$E_m = E_m + \rho(e_m - E_m) \quad (7)$$

using $\rho \in (0, 1]$ to weight the impact of new measurements. If no model quality E_m is above a threshold E_{min} , then a new model is created.

RL-CD, detailed in algorithm 4, integrates concept drift with single-objective RL. RL-CD maintains a set of models $\mathcal{M}$ from which it attempts to recognize previously seen concepts and to which it adds new models when no existing model is

⁴Because both [11] and [17] have the same abbreviation, neither are abbreviated here

adequate. The algorithm detects a concept change by assessing the reward prediction accuracy in terms of confidence and quality.

The confidence is the ratio of the number of steps that a model has been active and observed the given state and action, $N_m(s, a)$ to the most recent M experiences:

$$c_m(s, a) = \frac{N_m(s, a)}{M}. \quad (8)$$

The instantaneous quality is a weighted combination of two quality models, one for the transition change and one for the reward change. The quality equation is

$$e_m = c_m(s, a) (\Omega e_m^R + (1 - \Omega) e_m^T) \quad (9)$$

where

$$e_m^R = 1 - 2(Z_R(\Delta R_m)^2) \quad (10)$$

$$e_m^T = 1 - 2(Z_T \sum_{\kappa \in S} (\Delta T_m(\kappa))^2) \quad (11)$$

The Z terms are normalization factors. The weight between the transition model and reward model is Ω . The quality factors are used to compute a total quality for the model. The predicted reward update is

$$R_m(s, a) = R_m(s, a) + \Delta R_m \quad (12)$$

where ΔR_m is the moving average difference in the actual reward and the experienced reward at state s for action a :

$$\Delta R_m = \frac{r - R_m(s, a)}{N_m(s, a) + 1}. \quad (13)$$

The quantity $N_m(s, a)$ denotes the number of times action a was taken in state s for the model m . Since the memory is finite, N_m is bounded above by the length of the memory.

This method does not require assuming a baseline number of models and does not restrict the agent based on its spatial locality [60]. A limitation of this work is that it is model-based, depending on a transition function model. Further, Equation 13 has little tolerance for stochastic rewards.

Algorithm 4 RL-CD algorithm [60]

```

1:  $m_{cur} \leftarrow newmodel()$ 
2:  $\mathcal{M} \leftarrow m_{cur}$ 
3:  $s \leftarrow s_0$ 
4: repeat
5:   select  $a$  according to  $\pi_{m_{curr}}(s)$ 
6:   observe  $s'$ 
7:   observe  $r$ 
8:   for all  $m \in \mathcal{M}$  do
9:     Update  $E_m$  according to Eq 7
10:  end for
11:   $m_{cur} \leftarrow argmax_m(E_m)$ 
12:  if  $E_{m_{cur}} < E_{min}$  then
13:     $m_{cur} \leftarrow newmodel()$ 
14:     $\mathcal{M} \leftarrow \mathcal{M} \cup m_{cur}$ 
15:  end if
16:  Update  $T_{m_{cur}}$  as shown in [60] (Eq 1)
17:  Update  $R_{m_{cur}}$  according to Eq 12
18:   $N_m(s, a) \leftarrow \min(N_m(s, a) + 1, M)$ 
19:   $s \leftarrow s'$ 
20: until  $s$  is terminal

```

2.6.5 Adaptive Deep Learning. Deep Learning is a field that attempts to perform credit assignment to sub-components of a world model where sub-components are connected in a non-linear fashion. Deep Learning is typically discussed in the context of artificial neural networks [57] performing classification tasks. One type is the deep belief network [57], which has been modified for concept drift as the adaptive deep belief network (ADBN) [8]. While typical deep belief networks require a data set to permit re-training, the ADBN removes this requirement. The authors of [8] note that the ADBN is not able to explicitly forget a specific element, making it particularly susceptible to noise. The authors also note that the ADBN is sensitive

to the plasticity-stability dilemma but this can be tuned by adjusting the number of samples used to train between each epoch.

2.7 *Summary*

The challenges that reinforcement learning agents face in non-stationary environments are summarized in Section 2.2.2. Traditional reinforcement learning methods suffer from catastrophic forgetting and the plasticity-stability dilemma because they do not explicitly handle non-stationary environments. Of the few methods that do, they require models to be known, do not exploit multiple objectives, or otherwise are intractable in practical usage. This work seeks to fill these gaps.

The field of concept drift provides useful tools to assist multi-objective reinforcement learning in non-stationary environments by characterizing the environment using concepts. Through concept change detection as described in Section 2.5, a policy can be preserved for later re-use before it is impacted by the new environment. By using a running error estimate to recognize concepts, as described in Section 2.6.4.3, saved policies optimized under previous conditions may be reactivated when the environment returns to that concept and the concept is recognized. This mitigates catastrophic forgetting and provides faster reactivity for the agent. The learning rate may be changed to help manage the plasticity-stability dilemma by leveraging information from concept stability characterization.

Current approaches to integrating concept drift with reinforcement learning exploit anomalies in expected state transitions and in expected rewards. However, these approaches have a number of drawbacks: most are model based, most need a priori information in some form, and only one is multi-objective. This leaves an opening in the state of the art to contribute a multi-objective reinforcement learning (thus model-free) algorithm that explicitly addresses concept drift. The next section describes a virtually unexplored opportunity for additional information: feedback from multiple objectives.

III. Concept Drift-aware Multi-Objective Reinforcement Learning

This dissertation contributes concept drift-aware multi-objective reinforcement learning (CDMORL). CDMORL is a reinforcement learning (RL) algorithm that can characterize, detect change, and recognize concept drift in multi-objective reinforcement learning. CDMORL is designed with three methods to mitigate catastrophic forgetting and the plasticity-stability dilemma. This chapter also presents a novel integration of learning parameter decay with both re-initialization and recognition to allow convergence of behaviors even in non-stationary environments.

The algorithm developed integrates concept drift learners (CDL) with multi-objective reinforcement learning (MORL). It is based on multi-objective Q -Learning (MOQL) [69] (see Section 2.4.2), which was chosen as a minimalist approach to MORL and includes off-policy learning. This algorithm addresses the problem of catastrophic forgetting and the plasticity-stability dilemma in non-stationary environments.

The proposed algorithm manages a set of Q functions, policies π , and the history of residuals between the expected utility and the received reward to support concept characterization. This algorithm stores the old Q functions, policies, and residual history when the concept changes and re-loads them when the concept returns and is recognized. CDMORL evaluates the convergence of each sample (observation, action, resulting state, and resulting reward). If the sample indicates divergence, then a warning or drift is detected. The CDL thereby explicitly detects change and recognizes prior concepts, enabling MORL to mitigate catastrophic forgetting. This algorithm has the capability to activate a mitigation technique for the plasticity-stability dilemma, reducing the learn rate when the environment is characterized as unstable (or warning) and increasing it again when the environment is characterized as stable.

3.1 *Assumptions*

As with all concept drift learners, the algorithm proposed includes the following assumptions:

- changes do not occur more rapidly than the system can sample them, meaning that a “sufficient” signal-to-noise ratio exists. Specifically, the assumption is that concepts will be approximately stationary for a sufficient period of time¹. If an environment changes more quickly than the system, the system becomes chaotic [4].
- the number and the definition of concepts are not known in advance. Otherwise, the problem reduces to a set of stationary problems.
- the concept can be indirectly estimated from information provided to the agent.
- reward signals are provided at each step. Eliminating this requirement is a potential future research objective.

The convention in this dissertation is to minimize all objective functions.

3.2 *Symbols*

Table 2 summarizes the symbols used in the algorithms in this chapter. All parameters are based on either how many visits a state has received (exploration) or how many visits a state-action pair has received (everything else).

¹Consider the Nyquist sampling theorem: the change can be no more often than once every two samples

Table 2: Parameters and their nominal values. Parameters with * can be decayed. Specific values are discussed for each experiment.

Parameters used in <i>CDMORL</i>	
Symbol	Meaning
α^*	Learn rate. The undecayed rate is α_0 .
γ	Discount factor
δ_l	Policy learning rate while ‘losing’
δ_w	Policy learning rate while ‘winning’
ϵ^*	Probability of random action selection
θ_1^*	Threshold for a <i>warning</i>
θ_2^*	Threshold for <i>drift</i>
N_W	Window size
N_B	Batch size
T_{min}	Minimum steps for to retain a concept
S_{min}	Minimum converged state-action pairs to retain a concept

3.3 *CDMORL Algorithm Overview*

The CDMORL algorithm was informed by the error-signal-based approach in Appendix A and a statistical approach in Appendix B. The algorithm incorporates novel approaches to the concept characterization and recognition components, combining the CDL component tightly with the MORL component. If the characterization points to drift, it attempts to recognize a previous concept and reload it; if recognition fails, it creates a new concept. CDMORL is summarized in Figure 3 and Algorithm 5.

CDMORL makes several contributions:

1. unvisited state-action pairs cannot affect the concept: with this design, unvisited states (with randomly initialized Q values) are unable to impact concept classification. This contrasts with work such as [17, 60] which allowed unvisited states to impact concept estimation.
2. concept characterization is truly multi-objective: CDMORL analyzes each objective value independently and then characterizes the environment from that.

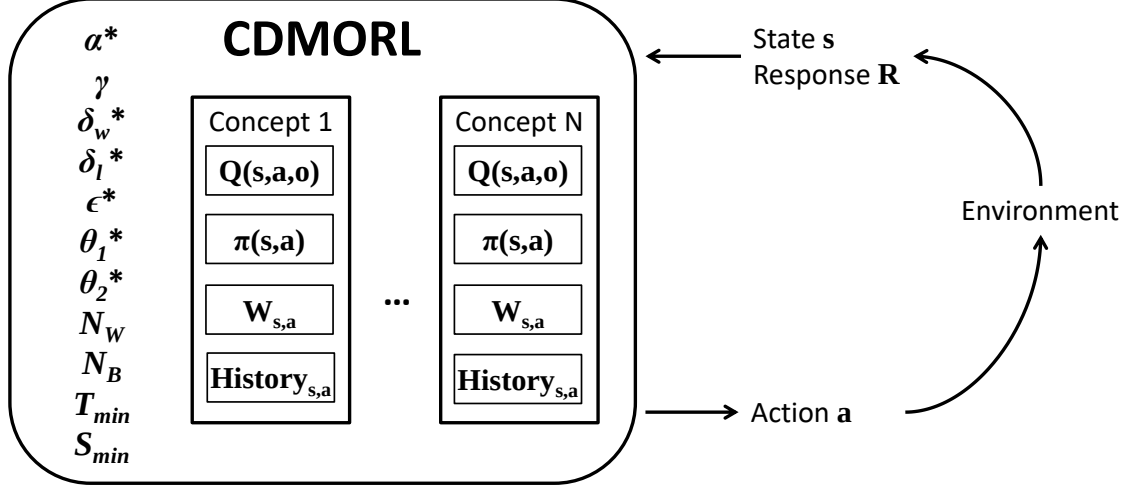


Figure 3: Overview of relationship among *CDMORL*, its constituent components, and the environment. Parameters with an asterisk are decayed in the agents with decay.

3. concept recognition attempts occur both at the time step when drifted and while initializing a new concept: concept recognition can happen any time the concept is not *stable* or *warning* (meaning *drift* or *initializing*), instead of only at the time step of need. Many algorithms only check at the moment drift is detected (such as [27]), though the opposite is that RL-CD checks at every time step [60]). This middle ground and utilization of an ‘initialization’ approach is, to the author’s knowledge, entirely novel.
4. concepts are not retained until they have either observed a specific number of state-action pairs (S_{min}) or until they have completed a specific number of steps (T_{min}). If a concept is the only concept, drift detections do not trigger creation of a new concept. If it is not the only concept and drift is detected or an old concept is recognized, the concept is deleted if it hasn’t met one of these thresholds.

Updates to the Q function includes a few modifications. First, the received reward is relative to the utopia point. If the utopia point is properly selected, this bounds the value of q to be greater than 0, so that the hypervolume calculation is correct. Hypervolume calculations do not include the utopia point as it has already

Algorithm 5 Concept Drift-aware Multi-Objective RL

```
1: initialize  $Q(s, a, o)$ ,  $\pi(s, a)$ 
2: repeat
3:   using policy  $\pi(s, a)$  with suitable exploration, take  $a$ 
4:   observe  $s$ ,  $\vec{r}$ 
5:    $stability \leftarrow \text{CharacterizeBatch}()$ 
6:   if  $stability$  is init or drift then
7:     if a previous concept has recurred then
8:       if current concept is not permanent then
9:         delete it
10:      end if
11:      load recognized concept
12:    else if  $stability$  is drift then
13:      initialize a new randomized concept
14:      change  $stability$  to initializing
15:    end if
16:  end if
17:  if  $stability$  is stable then
18:    update saved concept definition
19:  end if
20:  if using WoLF then
21:    adjust learn rates per WoLF
22:  end if
23:  if using decaying learn rates then
24:    decay learn rates for current concept
25:  end if
26:  for each objective  $o$  do
27:     $Q(s, a, o) \leftarrow Q(s, a, o) + \alpha(s, a) ((\vec{r}_o - z_o^*) + \gamma \max_{a'} Q(s', a', o) - Q(s, a, o))$ 
28:  end for
29:  learn  $\pi$ 
30: until terminal
```

been accounted for in updating q . Next, the agent selects the action yielding the maximum value based only on the independent Q table for that objective, regardless of the action selection strategy².

This section goes into the new method in greater detail. First, concept characterization, change detection, and recognition are covered, discussing the batch approach

²Even though the agent is a minimizing agent, maximum is the operation used. The agent then knows what to avoid.

to characterize a batch and the convergence methods to characterize a sample. Then, how this feedback is exploited by the MORL component is detailed, notably concept recognition, concept definition, managing the plasticity-stability dilemma, and decay.

3.4 Concept Characterization, Change Detection, and Recognition

Concept characterization, change detection, and recognition informs the MORL agent. These functions are performed by a concept drift learner (CDL). Each CDL is associated with a Q -function, a policy π , and the history of residuals for each state-action pair. The tracker consists of a window of samples for each state-action pair, $W_{s,a}$, with each window containing a maximum of N_W samples in its history. In CDMORL, the active CDL monitors the residual between the expected and actual reward, then labels this data stream stability as *stable*, *warning*, or *drifted*. The CDL classifies states with insufficient experience as *initializing*. To be persisted, a model must have previously achieved *stable* in a pre-set number of state-action pairs. The active concept learns every time after recognition and re-initialization have occurred, guaranteeing that a *drifted* concept is not active. Batches consist of several samples that vote to determine the concept.

3.4.1 Batch-based CDMORL. Noise from random inputs can easily cause false drift detections and warnings. One way that concept drift mitigates this problem is through batches that conduct winner-take-all votes. Instead of a clear-cut winner, the batch forms the basis of a new concept. While MORL progresses, samples $(s, a, \vec{r})$ are saved until a batch reaches capacity N_B . Then, all samples are processed at the same time, resulting in a stability vote for that sample. All concepts, including stored concepts, process the batch.

Batches are scored with a point value, which determines the characterization. Votes of *initializing* don't count, but if more than half of the votes are for *initializing*, that is the assessment. Otherwise, a vote of *drift* is -1, *warn* is 0, and *stable* is 1. The total vote score is divided by the number of non-*initializing* samples. A score of $-1/2$

or less is characterized as *drift*, a score greater than $1/2$ is characterized as *stable*, otherwise the concept is *warn*. This voting scheme works with a batch of any size one or greater. Algorithm 6 outlines this process, with the voting mechanism described in Algorithm 7.

Algorithm 6 Characterize Batch

```

1: function CHARACTERIZE BATCH(Batch of  $s, a, \vec{r}$ )
2:   for each sample  $s, a, \vec{r}$  do
3:      $labelList \leftarrow CharacterizeSample()$ 
4:   end for
5:   if  $voteList$  is mostly initializing then
6:      $label \leftarrow initializing$ 
7:   else
8:      $voteTotal = 0$ 
9:     for each  $vote$  in  $vote\ list$  do
10:      if  $label$  is drifted then
11:         $voteTotal \leftarrow voteTotal - 1$ 
12:      else if  $label$  is stable then
13:         $voteTotal \leftarrow voteTotal + 1$ 
14:      end if
15:    end for
16:     $label \leftarrow Tally\ Vote(voteTotal)$ 
17:  end if
18:  return  $label$ 
19: end function

```

Algorithm 7 Tally Vote

```

1: function TALLY VOTE( $vote\ list$ )
2:    $aveVote \leftarrow voteTotal / (non\ Init\ labels)$ 
3:   if  $aveVote \leq -1/2$  then
4:      $label \leftarrow drifted$ 
5:   else if  $aveVote \leq 1/2$  then
6:      $label \leftarrow warning$ 
7:   else
8:      $label \leftarrow stable$ 
9:   end if
10:  return  $label$ 
11: end function

```

3.4.2 Characterizing Samples. Sample characterization compares, for the specified state-action pair, the current residual against previous residuals. Each con-

cept maintains a window of samples $W_{s,a}$ of the residual between the expected utility $\vec{q}_c$ and observed reward $\vec{r}$, that contains between zero and N_W samples (discarding oldest). Each sample w is a residual, and w_o is the residual for a specific objective o . For each objective, the algorithm finds the ratio between the current residual and the most similar residual stored in the history window.

Algorithm 8 Characterize Sample

```

1: function CHARACTERIZE SAMPLE( $s, a, \vec{q}_{c,s,a}, \vec{r}$ )
2:    $residual \leftarrow \vec{q}_{c,s,a} - \vec{r}_{s,a}$ 
3:   if residual history of  $s, a$  is empty then
4:      $label \leftarrow init$ 
5:   else
6:     Find the minimum proportion change for each objective:
7:      $voteTotal \leftarrow 0$ 
8:     for objective  $o$  do
9:       for sample  $w$  do
10:         $result \leftarrow abs(1 - residual_o/w_o)$ 
11:         $ratio_o \leftarrow \min(result, past\ results)$ 
12:      end for
13:      if  $ratio_o < \theta_1$  then
14:        (This is a vote for stable)
15:         $voteTotal \leftarrow voteTotal + 1$ 
16:      else if  $ratio_o \geq \theta_2$  then
17:        (This is a vote for drift)
18:         $voteTotal \leftarrow voteTotal - 1$ 
19:      end if
20:    end for
21:     $label \leftarrow TallyVote(voteTotal)$ 
22:    if Residual has never converged to stable before, counting this step then
23:       $label \leftarrow init$ 
24:    end if
25:  end if
26:  add  $residual$  as  $w_{s,a}$  to history for  $s, a$ 
27:  return  $label$ 
28: end function

```

3.5 Integrating the CDL into MORL

The MORL algorithm proceeds as though the environment is approximately stationary, providing the CDL and all archived CDLs with the information they need

each step. This section discusses how the MORL algorithm reacts to CDL detections of *initializing*, *warning*, and *drift*. The intervention for mitigating catastrophic forgetting is to store concepts and attempt to recognize concepts during drift and initialization. This is a core feature of all versions of CDMORL. Three additional interventions are experimentally evaluated to improve concept characterization (required for catastrophic forgetting mitigation) and to mitigate the plasticity-stability dilemma: the use of batches (described above, used to improve precision for concept characterization, which in turn impacts catastrophic forgetting), the use of decaying parameters, and the use of a variable learn rate.

3.5.1 Mitigating Catastrophic Forgetting. Catastrophic forgetting occurs when an agent has previously learned a behavior for a given concept, only to return to it after drift and not recall this solution. This requires storing the knowledge and a way to recall it later. Recognition generally follows the RCD algorithm [27] (see Section 2.5.6). Concepts must be *stable* or have a *warning* characterization to be reloaded, with *stable* concepts preferred to *warning* concepts³. To accomplish this, all concepts continually characterize the environment, to check for matches.

Having too many overlapping concepts presents a problem for concept recognition, because multiple stable classifiers compete to be recognized, and impede the rate at which the agent can learn its environment. To protect concept definitions, concept definitions only update while stable. To prevent an explosion of classifiers, *CDMORL* proactively deletes classifiers created in response to noise. That is, if a classifier is initializing, a recognition occurs, and the classifier has been stable for fewer than 100 steps, the algorithm deletes the initializing classifier as a defense against noise and unbounded memory consumption.

3.5.2 Mitigating the Plasticity-Stability Dilemma. The plasticity-stability dilemma is the balance between an agent learning too rapidly and overfitting to noise,

³Ties, which are uncommon when using batches, are broken first by highest vote, then arbitrarily.

or learning too slowly and failing to adapt. To manage this, *CDMORL* uses the WoLF algorithm (Section 2.6.3). This adjusts learn rates when the concept characterization is *warn*, opting to learn more conservatively while the agent is doing well.

3.5.3 Decay. Exploration rates ϵ , policy learn rates δ_{losing} and $\delta_{winning}$, and Q learn rate α are all subject to decay, based on the number of steps t the concept has been active. Thus, newly initialized concepts start with fresh rates, while recognized concepts are considered more mature, and have lower rates to facilitate convergence. While this was listed as a primary culprit of the stability-side of the plasticity-stability dilemma, the argument is mitigated when decay occurs within a stable environment, that is, within a single concept. Because of that, decay is expected to improve agent performance.

To prevent overlap resulting in false recognitions or duplicate policies, new concepts are initialized with a randomized Q function, a step count t of 0 (meaning no rate decay yet), and a randomized behavior policy⁴.

3.6 Summary

The *CDMORL* algorithm uses convergence measures for each individual objective function and introduces an *initializing* characterization. The second algorithm has a number of interventions which are explored experimentally, specifically a variable learning rate; the use of learning and exploration rate decay; and whether batches should be used.

The batch intervention is designed to improve classifier precision at the expense of a minor delay in learning, needed to mitigate both catastrophic forgetting and the plasticity-stability dilemma. The WoLF learn rate intervention is designed to mitigate the plasticity-stability dilemma, while learn rate and exploration decay is intended

⁴These randomizations may not be necessary, so long as the windows are reset, but are left for future work.

to help mitigate the plasticity-stability dilemma but indirectly helps with accurate concept definitions, making it easier to recognize concepts correctly.

Chapter 4 overviews a testing plan to demonstrate these improvements. Additional extensive treatment is given in Appendix C.

IV. Experimental Evaluation

Agents operating in the real world are expected to optimize multiple objectives while coping with non-stationarity in the form of concept drift. This chapter describes the approach to testing whether concept drift-aware multi-objective reinforcement learning (CDMORL) is more accurate than multi-objective reinforcement learning (MORL). Specifically:

1. Does CDMORL have greater accuracy than MORL, with respect to hypervolume? (Accuracy is defined below.)
2. Which concept drift handling techniques are most accurate, with respect to hypervolume?
3. Which multi-objective concept drift strategy most accurately identifies the correct number of concept changes?
4. Which multi-objective concept drift strategy most accurately identifies the correct number of concepts?

The first section of this chapter presents methods to objectively measure the performance of a MORL agent and its concept drift learner (CDL) in non-stationary environments. The second section summarizes the nomenclature for the algorithms tested herein. The third and fourth sections present two test domains for multi-objective non-stationary research, graph traversal in MERLIN and resource gathering. The chapter ends with a summary of the experimental evaluation approach.

4.1 *Measurement*

The domain of multi-objective reinforcement learning under concept drift is, to the best of the author’s knowledge, a novel test domain. Some similar work has been done for non-stationary evolutionary algorithms, called evolutionary dynamic optimization (EDO) algorithms [13, 48], and where practical that work has been adapted here. This section describes two approaches to measure agent performance, a concept change detection measure, and a concept recognition measure. The first part of

this section covers agent performance evaluation methods, and the second part covers measures related to concepts.

4.1.1 Agent Performance Measures. In traditional reinforcement learning, agent performance is often measured by examining the cumulative reward (possibly divided by the number of steps taken) or the instantaneous reward. From either of these approaches, it is possible to produce a final result, whether by looking at the end cumulative reward or the sum of instantaneous rewards. The use of multiple objectives makes it difficult to declare a single winner, if multiple solutions have membership on the Pareto front (see Section 2.3). However, concept changes effectively change the environment, making either final measure less meaningful. To address these two issues, two measures are used: **instantaneous hypervolume** and **accuracy**. These measures are adapted from multi objective optimization evolutionary dynamic optimization (EDO) (see Section 2.6.1) measures.

4.1.1.1 Instantaneous Hypervolume. Instantaneous hypervolume scalarizes the reward received by the agent at each step with the hypervolume indicator. The hypervolume indicator avoids biasing the agent toward one objective or the other.

The instantaneous hypervolume is useful to visually observe performance changes on concept changes. In an experiment with concept drift, all agents should have a visible, sharp change to instantaneous performance when the concept changes. Agents that are able to adapt, whether from detecting change or from having high plasticity, should achieve a visible improvement between changes, thereby creating an uneven sawtooth pattern. Agents unable to adapt will experience performance differences, for better or for worse, by behaving as though the environment did not change. Agents that recognize concept change should illustrate a quick recovery - rather than gradually adapting, they should quickly jump to a previously established solution.

Because the convention in this dissertation is to minimize, lower values of instantaneous hypervolume are better than higher values.

4.1.1.2 *Accuracy.* Accuracy¹ summarizes, as a function of time step t , performance as a scalar. Performance compares all agents within an experiment. When evaluating agent i , up to two other agents j and k , $j \neq k$ may be necessary to compute accuracy, because the measure is relative within a set. At step t , agent j has the greatest average instantaneous hypervolume $\bar{F}_j(t)$ and agent k is the agent with the least average instantaneous hypervolume. Due to the way they are defined, the agents designated j and k may change each time step. Agent i may be the same as either agent j or agent k , or different. Accuracy for agent i is computed as

$$accuracy(i, t) = \frac{\max_j(\bar{F}_j(t)) - \bar{F}_i(t)}{\max_j(\bar{F}_j(t)) - \min_k(\bar{F}_k(t))} \quad (14)$$

thus giving agents with the lowest instantaneous hypervolume (which, since all objectives are minimized, makes it the best agent at that time step) the highest accuracy, and agents with the highest (worst) instantaneous hypervolume get the lowest score.²

Accuracy was selected because it integrates performance over time, making graphs easier to read. Accuracy, when computed for the final step, presents a single scalar for each agent design, making it easier to contrast performances. Some notes about accuracy:

- A greater accuracy value outperforms a lesser accuracy value.
- The accuracy measure assumes all algorithms are run in the same environment settings, meaning results are not comparable across different experiments.
- Accuracy accounts for concept changes.

The combination of instantaneous hypervolume with accuracy gives insight both into the overall step-by-step performance of an algorithm as well as an overall final assessment.

¹Accuracy is loosely based on the measure by the same name from EDO [13]. It characterizes the performance of an agent with respect to its peers in a given experiment configuration.

²The values of *Max* and *Min* in this dissertation sampled from the last 90% of data, because the first 10% tended to skew the scaling of the graph, which made visual inspection and analysis unnecessarily hard.

4.1.2 Classifier Performance. Concept drift classifier measures answer the questions of how well the agent can reliably detect change and how well it recognizes other concepts. Two measures answer these questions: **false alarm count** and **final concept count**.

4.1.2.1 False Alarm Count. The *false alarm* count quantifies how often the concept classifier detected concept drift, but no such drift was present. Change detection inevitably suffers latency between change and detection, which makes it challenging to discern between a false positive and a delayed correct change. However, if the precise number of concept changes is known, then the lower bound on the number of false positives can be computed exactly.³

A high false alarm count indicates that a classifier is unreliable in detecting drift. In CDMORL, this means it will compensate for non-existent problems, which will in turn lead to convergence challenges and possibly cause catastrophic forgetting.

4.1.2.2 Final Concept Count. The question of whether a concept is recognized can be bounded by counting the number of concepts an agent is managing at the end of the experiment, if the true number of concepts is known. The *final concept count* indicates how many concepts were detected and finalized by the classifier (if the current concept at the end is still initializing, it is included in the final concept count). Because agents can achieve recognitions while initializing a new concept, this allows it to discard a duplicative concept and restore previous information (note that this results in two concept change detections).

The final concept count shows how often well the agent performs recognitions. If there are too few concepts in the final concept count, then either the agent finalized concepts before old concepts could be recognized, or the agent overreacted to noise. If too few concepts are in the final concept count, it is possible that the experiment

³This also means that the upper bound on the number of true positives can be computed, but this is unhelpful in later analysis as almost every algorithm that detects change detected more changes than there were actual concept changes, meaning every algorithm has the same upper bound.

did not present sufficiently meaningful change that the agent detected change. It is also possible that the agent is too plastic, which means that catastrophic forgetting will occur as the agent overwrites information repeatedly.

A final concept count that is too high guarantees that an agent is unpredictable at achieving concept recognitions. A final concept count that is too low is possibly indicative that the agent is unable to discern between different concepts.

4.1.2.3 Examining for Statistical Significance. For both the false positive count and the final concept count, statistical significance testing is performed using the Kruskal Wallis H test [39]. The significance level selected is 0.05. Tests compare types of agents within a single experiment, pooling the data from all types together. They are compared against the remaining tests in that category.

4.2 Algorithm Nomenclature

The CDMORL algorithm underwent four designs, as shown in Chapter III, Appendix A, and Appendix B (a fourth design, using a weighted vote version of Chapter III is not documented here as it was a poor approach and overreacted to noise). Chapter III presented several different techniques to mitigate concept drift, which can be combined. The naming scheme for the different combinations is presented in Table 3. The algorithms are presented in four groups:

- Concept Drift-aware Multi-Objective Reinforcement Learning;
- Concept Drift-aware Multi-Objective Reinforcement Learning with batches;
- Concept Drift-aware Single-Objective Reinforcement Learning; and
- Reference Reinforcement Learning algorithms.

Multi-objective Reinforcement Learning (MORL) algorithms contain “MORL”. Algorithms with a concept drift learner contain “CD”. Algorithms that process data in batches have the prefix *b-*. Algorithms with decay use the suffix *-d*, while algorithms using WoLF use the suffix *-w*.

Single-objective reinforcement learning algorithms start with *SO*. All single objective agents make decisions using a single *Q*-table, that learns on rewards scalarized by Hypervolume. The agents with *_MC* use multiple objectives to track concept drift, while *_SC* agents use a single objective to track concept drift scalarized with hypervolume. For the *_MC* agents, the agent tracks multiple *Q*-tables for each objective, plus the *Q*-table it uses for learning the environment.

Table 3: Summary of algorithms.

Name	Objectives	Detects Drift?	WoLF	Decay	Batches
Concept Drift-aware Multi-Objective Reinforcement Learning					
CDMORL	multiple	yes	no	no	no
CDMORL-d	multiple	yes	no	yes	no
CDMORL-w	multiple	yes	yes	no	no
CDMORL-d-w	multiple	yes	yes	yes	no
CDMORL with Batches					
b-CDMORL	multiple	yes	no	no	yes
b-CDMORL-d	multiple	yes	no	yes	yes
b-CDMORL-w	multiple	yes	yes	no	yes
b-CDMORL-d-w	multiple	yes	yes	yes	yes
Single Objective RL with Concept Drift Detection					
SO_SC	single	scalarized	no	no	no
b-SO_SC	single	scalarized	no	no	yes
SO_MC	single	yes	no	no	no
b-SO_MC	single	yes	no	no	yes
Reference Algorithms					
MORL	multiple	no	n/a	no	n/a
MORL-d	multiple	no	n/a	yes	n/a
SO	single	no	no	no	no

Since there are 15 algorithms for each experiment, for clarity, plots of results contain all algorithms from one of the first three categories plus the reference algorithms.

4.3 Experiment Design and Variables

Three different sets of experiments with six instantiations each (eighteen total) are used to empirically answer the four research questions at the start of this chapter. The rest of this section identifies the independent and dependent variables.

4.3.1 Independent Variables. The independent variables in these experiments are

- Environment design (18 total). The chief purpose behind the different designs is to evaluate CDMORL in different instantiations and contexts, to reduce the likelihood of a anomalous result. The specifics behind these experiments is explained in the next section.
- Algorithm design (15 total designs). For each experiment, all 15 agent designs are evaluated. These agents are listed in Table 3.

Thirty trials are conducted for each agent for each experiment, and the results are analyzed using the measures in Section 4.1.

4.3.2 Dependent Variables. The dependent variables in this work are

- instantaneous hypervolume (lower is better)
- accuracy (higher is better; best possible value is 1.0, worst possible value is 0.0)
- false alarm count (lower is better; directly derivable from the presented concept change count; the correct number, which is given with the table, is either 0 or 15, and anything in excess is a false alarm)
- final concept count (closer to the correct number is better; the correct number is given with the table, but in this work, is always either 1 or 4)

4.4 Environments

This section describes the different experiment designs. All three experiments are based on the MERLIN [15] tool. MERLIN is a multi-objective Markov decision problem (MOMDP) generator [25] used to generate a set of directed graphs with a parameterizable number of states, actions, objectives, and distributions for objectives⁴. The graphs have no unreachable nodes and no dead ends. The nodes in the

⁴MERLIN is available at <https://github.com/deong/merlin> and was retrieved October 12, 2015.

graph are states, the connections are possible actions, and each action yields rewards. By default, MERLIN test cases have discrete states, deterministic transitions within each concept, and invariant rewards. These graphs are the basis of three experiment designs:

1. Stationary MERLIN: The first set of experiments, #1-#6, primarily serves to study CDMORL in the absence of drift. This experiment uses the information from MERLIN without modification, and is thus stationary. The MERLIN domain consists of randomly generated MOMDPs with discrete states, deterministic transitions, and invariant rewards.
2. Non-Stationary MERLIN: The second set of experiments, #7-#12, studies CDMORL in environments with concept drift where every state and action gives partially observable information as to the current concept. It cycles through four different graphs, which is the source of concept drift.
3. Resource Gathering: The third set of experiments, #13-#18, studies CDMORL in an environment where only a few states and actions give information as to the current concept, with some information being given stochastically. The final experiment uses substantial modification, and is called resource gathering [3]. Resource gathering is representative of a strategy game, and features a stochastic environment with non-deterministic transitions and rewards that vary by as much as two orders of magnitude.

These three experiment types are explained in detail in the rest of this section.

4.4.1 Stationary MERLIN. The Stationary MERLIN experiments have one concept per experiment, where each concept is defined by a single graph. Graphs contained 25 states with 4 actions and 3 objectives. Each action leads to a state. Rewards, which are of constant value, are normally distributed according to $\mu = 0, \sigma = 1$ and are forced to not exceed $[-3, 3]$. The following script generated the

MDPs:

```
python merlin.py -t discrete --graph-type random -n 25 -m 4 -k 3.
```

The parameters used in CDMORL for these experiments is shown in Table 4. These parameters reflect the deterministic nature of MERLIN: deterministic transitions and deterministic rewards, resulting in perfectly predictable concept changes, thereby minimizing the need to account for noise whether in windows or in batches.

Table 4: Parameters and their values for MERLIN.

Symbol	Default	Meaning
α^*	0.2	Learn rate. When decayed, approaches 0.1.
γ	0.9	Future discount
δ_l	0.2	Policy learning rate while ‘losing’. Default when WoLF disabled.
δ_w	0.05	Policy learning rate while ‘winning’
ϵ^*	0.3	Probability of random action. When decayed, approaches 0.1.
θ_1^*	$1.0\alpha^*$	Threshold for a <i>warning</i>
θ_2^*	$3.0\alpha^*$	Threshold for <i>drift</i>
N_W	1	Window size
N_B	3	Batch size
T_{min}	100	Minimum steps to be permanent
S_{min}	10	Minimum converged state-action pairs to be permanent

4.4.2 Non-stationary MERLIN. This problem builds on the approach of Stationary MERLIN. Concept change is induced by changing the transition function every 1000 steps by rotating which action goes to which node (state). In the first concept, action 1 a_1 transitions to state s_1 and action 4 transitions to state 4 ($a_4 \rightarrow s_4$). In the second concept, this rotates, so that $a_1 \rightarrow s_2$ and $a_4 \rightarrow s_1$, and so on. Because each graph has the same number of states and actions, the environment can move seamlessly from one concept to the next. The concept changes in MERLIN impact the agent instantaneously. This problem uses the same parameters as Stationary Merlin, as shown in Table 4.

4.4.3 Resource Gathering. The Resource Gathering (RG) domain is based on the version presented by Barrett and Narayanan [3], which is a 5x5 grid world featuring two enemies and two treasures. Every time the agent enters a square with an enemy, there is a chance the agent is defeated, and sent back to the start empty-handed. Every time the agent enters a square with a treasure, the agent collects the treasure, thereby making it unavailable until the problem resets. A reset happens when the agent returns to the square it started in. The agent must minimize times the enemies defeat it and maximize how often it collects and brings back each treasure, with each treasure serving as an objective. The domain is an abstraction of a strategy game.

The RG domain is used here, though with modification: as presented here, the RG domain is built on top of MERLIN, so instead of a grid, RG uses the same directed graph format. Here, the problem has 36 states instead of 25 and 2 objectives instead of 3 (it still has 4 actions). The objectives are to minimize damage taken and debt, which are both incurred in small amounts each step. Within the maze are three enemies and two treasures, placed at pre-specified locations that are the same for all experiments. Each enemy has a 40% chance to defeat the agent, sending it back to start with 100 damage. If the agent isn't defeated by an enemy, it takes a small amount of damage and debt. Treasures are cashed in immediately, yielding immediate -50 to debt. However, they can only be picked up once, then the agent must return to the start node before they re-appear and can be collected again.

For steps that don't contain treasure or where the enemy doesn't defeat the agent, a reward is still given⁵. The mean value for the rewards $R(s, a)$ (being damage and debt) is normally distributed with $\mu = 3, \sigma = 1, \in [0, 6]$, and when the agent receives the reward, it has Gaussian noise applied with a standard deviation of 0.15, not to exceed three standard deviations. The utopia point is $z^* = (-1, -75)$ ⁶.

⁵What to do with rewards of 0 is a limitation of this work and is a topic for future engineering.

⁶In the code, the treasure is worth 70 points instead of 50 if it is co-located with an enemy, however, that is never the case in the experiments in this dissertation.

Concept changes alter the location of the enemies and treasures. However, the transition function otherwise does not change: $a_1 \rightarrow s_1$ always holds, except when the agent is defeated by an enemy, then it is teleported back to the starting space. Parameters are shown in Table 5. Concept duration is 2500 steps.

Table 5: Parameters and their nominal values for resource gathering.

Parameters used in <i>CDMORL</i>		
Symbol	Default	Meaning
α^*	0.2	Learn rate. When decayed, approaches 0.1.
γ	0.3	Future discount
δ_l	0.2	Policy learning rate while ‘losing’. Default when WoLF disabled.
δ_w	0.05	Policy learning rate while ‘winning’
ϵ^*	0.3	Probability of random action. When decayed, approaches 0.1.
θ_1^*	$1.0\alpha^*$	Threshold for a <i>warning</i>
θ_2^*	$3.0\alpha^*$	Threshold for <i>drift</i>
N_W	7	Window size
N_B	10	Batch size
T_{min}	300	Minimum steps to be permanent
S_{min}	50	Minimum converged state-action pairs to be permanent

The RG domain is more challenging than MERLIN in several ways:

- RG yields stochastic rewards. The CDMORL agent is designed to identify drift based on convergence, and so this makes convergence more challenging.
- Except for the enemies and treasures, $R(s, a)$ does not change between concepts: in MERLIN, this is how the agent tracked drift, whereas here it is not as useful but instead is noise.
- The feedback from the enemies is non-deterministic, and so landing in the same square as an enemy results in a non-deterministic state transition.
- Both being killed by an enemy and capturing a treasure results in a reward that is an order of magnitude or more greater than the value of the normal signal from the transition a in s .

4.5 *Summary*

The experiments provide four measures of interest: instantaneous hypervolume, accuracy, false alarm count, and final concept count. These measures, when analyzed across the three types of experiments, are expected to demonstrate that CDMORL has consistently better performance than MORL.

V. Experiment Results

This chapter describes the results of the experiments. The first three sections describe the experiments performed and provide summaries of each measure. Accordingly, the next three sections present the results of stationary MERLIN, non-stationary MERLIN, and resource gathering (RG), respectively. For these sections, the first three subsections show graphs of the average *instantaneous hypervolume* and *accuracy* by category (CDMORL, CDMORL with batches, and concept drift-aware single objective RL). The fourth subsection is a summary of agent performance, while the fifth and sixth subsections summarize *false positive count* and *final concept count*. The last section of this chapter presents overall discussion and analysis of the results, with the first part aligned to the four research questions at the start of Chapter IV and the second part has general discussion of other observations and thoughts.

5.1 Stationary MERLIN Tests

The six stationary MERLIN tests serve as control tests to confirm that CDMORL is as accurate in terms of instantaneous hypervolume as existing non-drift algorithms. Since the results for all six experiments are similar, only one experiment result is shown. The complete set of results are reported in Appendix C. Figures 4 through 6 show the results of the stationary portion of tests on a representative test. This section also reports the concept change count and final concept count.

5.1.1 Accuracy Test: CDMORL Agents. Figure 4 is a plot of the results of CDMORL with and without decay, and with and without WoLF. It contrasts performance with the two multi-objective reference algorithms, MORL and MORL-d. These three graphs are representative of the other five stationary experiments. Generally, in all six experiments, the non-decaying algorithms performed very close to the baseline MORL algorithm. The MORL-d algorithm, which is reasonably expected to perform the best as it has no chance of catastrophic forgetting in a stationary environment, typically performed in the best three accuracies (this is an atypical result for MORL-d).

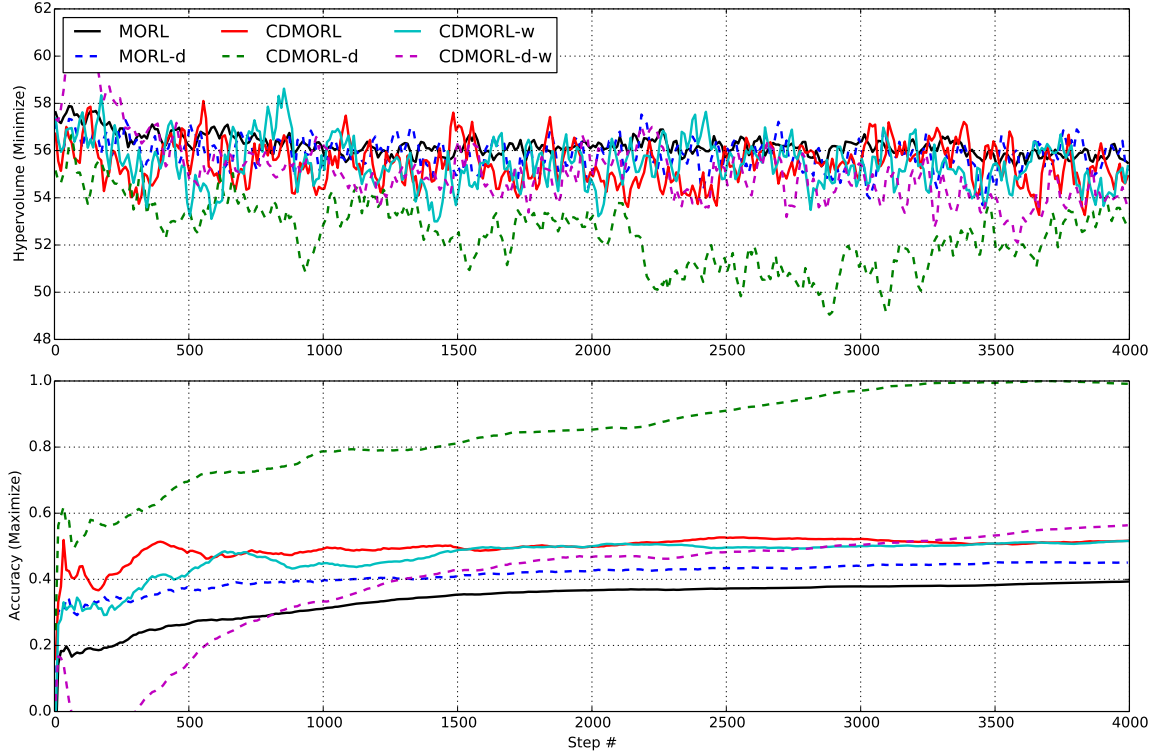


Figure 4: Hypervolume and accuracy for stationary MERLIN experiment 4, comparison among non-Batch agents and reference agents.

5.1.2 Accuracy Test: Batch-based Agents. Batch-based CDMORL algorithms are shown in Figure 5. The batch-based algorithms with decay generally achieved a similar accuracy as the non-batch based CDMORL algorithms with decay. The batch-based algorithms without decay performed about the same as the CDMORL algorithms without decay did. Experiment 6 is the exception to this. This occurrence is more characteristic of the MERLIN environment, which makes it easy to get trapped in one portion of the state space, than it is of the algorithms. While b-CDMORL-d-w was outperformed by b-CDMORL-d on four experiments, it outperformed the baseline algorithms more times (five compared to four).

5.1.3 Accuracy Test: Single Objective Agents. The single objective reinforcement learning (SORL) algorithms generally had unremarkable performance on the stationary set in contrast with the MORL algorithms, achieving a greater accuracy on Experiments #2 and #4 and generally worse on the rest. For these stationary

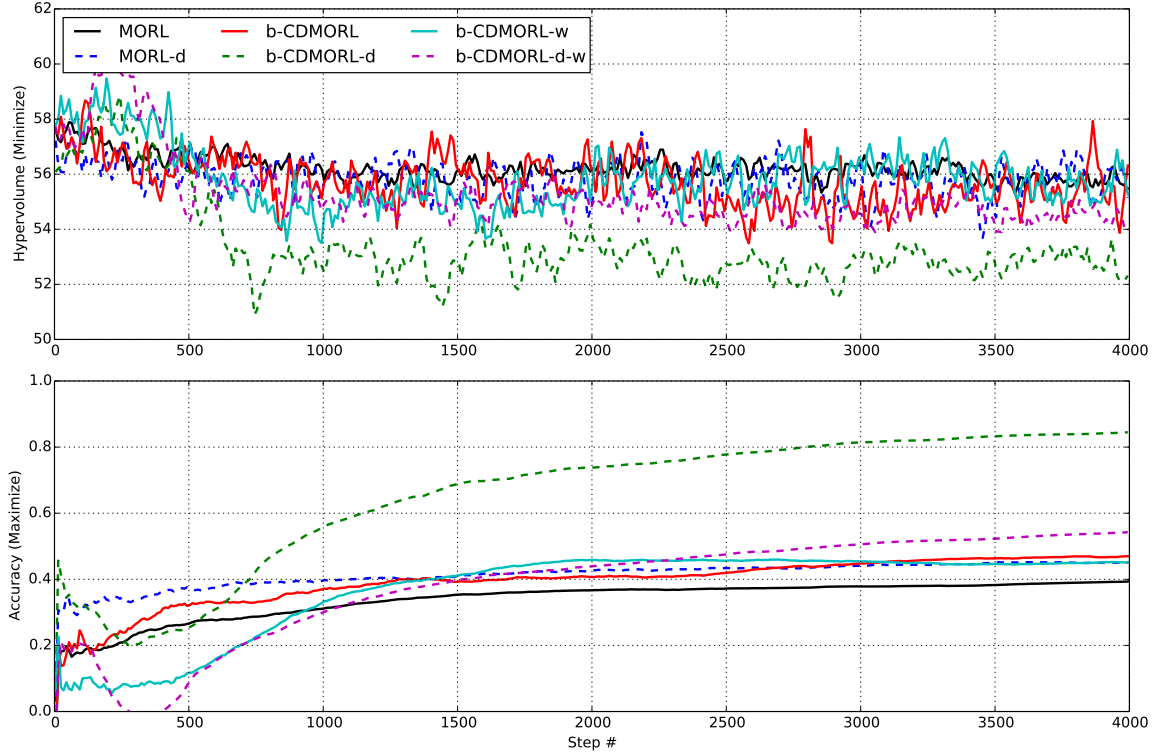


Figure 5: Hypervolume and accuracy for stationary MERLIN experiment 4, comparison among Batch agents and reference agents.

experiments, the agents using multiple objectives to detect drift ($_MC$ algorithms) performed slightly better than those using a scalarized drift detection method ($_SC$). Overall, the relative performance between the drift-detecting algorithms and the non-drift detecting algorithm, since no algorithms used decay or WoLF, is that the SO algorithm presumably got stuck in sub-optimal positions, such that it was better for the drift-detecting algorithms to erroneously detect drift and thus create a new agent than it was to remain in the local optimal found by SO.

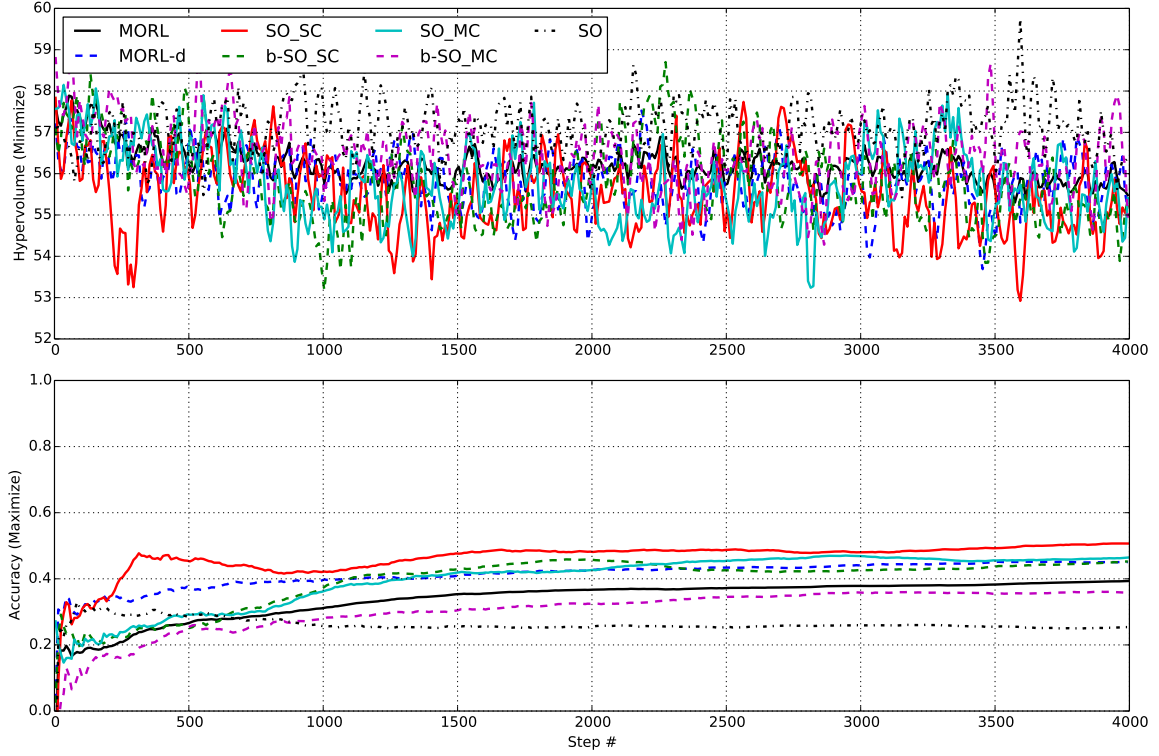


Figure 6: Hypervolume and accuracy for stationary MERLIN experiment 4, comparison among single objective and reference agents.

5.1.4 *Summary of Accuracy Tests in Stationary MERLIN.* Table 6 summarizes the final accuracy for each algorithm, which shows a consistently greater accuracy for CDMORL algorithms using decay ($-d$) than was achieved by the reference algorithms. The batch-based algorithms (b -) performed about the same as the non-batch-based CDMORL algorithms. *MORL-d* nominally should perform the best because it isn't looking for non-existent concept changes, and so it ought to converge to a reasonable solution set and stay there. However, the solution it converged to was typically poor.

The likely reason that the CDMORL algorithms performed better, when parity was expected, is because the γ term in the Q update could trigger a change in concept characterization when a subsequent state was discovered to have particularly poor performance. A characterization of drift, while wrong, could trigger creation of a new agent to explore a new region of the state space with renewed decay terms. A characterization of unstable, while also wrong, could prevent over-reaction to significant γ terms.

Overall, the CDMORL algorithms using decay performed the best, followed by MORL-d and then the CDMORL algorithms not using decay. These algorithms were followed by MORL and the single objective algorithms that have drift detection, followed last by the single objective RL reference algorithm.

Table 6: Summary of ending accuracy for stationary MERLIN. For each experiment, the best algorithm in each category is **bold**. The overall best is also *italicized*.

Algorithm	Experiment Number					
	1	2	3	4	5	6
CDMORL						
CDMORL	0.550	0.448	0.268	0.516	0.283	0.494
CDMORL-d	0.829	0.961	0.956	0.991	0.962	0.895
CDMORL-w	0.479	0.484	0.287	0.516	0.227	0.461
CDMORL-d-w	0.649	1.000	0.669	0.564	0.976	1.000
Batch-based CDMORL						
b-CDMORL	0.594	0.505	0.296	0.469	0.217	0.370
b-CDMORL-d	1.000	0.923	0.315	0.845	0.846	0.325
b-CDMORL-w	0.546	0.455	0.152	0.452	0.258	0.391
b-CDMORL-d-w	0.843	0.732	0.986	0.543	1.000	0.000
Single Objective Drift-Aware RL						
SO_SC	0.153	0.402	0.160	0.507	0.262	0.358
b-SO_SC	0.105	0.371	0.176	0.452	0.151	0.489
SO_MC	0.149	0.464	0.265	0.464	0.264	0.451
b-SO_MC	0.112	0.438	0.224	0.359	0.206	0.434
Reference						
MORL	0.257	0.432	0.242	0.393	0.305	0.465
MORL-d	0.737	0.047	0.592	0.451	0.679	0.404
SO	0.003	0.129	0.080	0.254	0.175	0.208

5.1.5 Test for Correct Concept Change Count in Stationary MERLIN.

Table 7 counts the average number of concept changes each classifier detected. Because there were zero concept changes, all reports are false positives. The batch-based algorithms with decay typically identified the fewest changes, distantly followed by the batch based algorithms without decay, the batch-based single objective RL algorithms, and the CDMORL algorithms with decay. The third tier of performance are the two CDMORL algorithms with neither batches nor decay. Each false change prevented the correct concept from converging as quickly as it could have. It appears that a limited number of fresh starts ultimately helped the overall performance, because the four algorithms with decay all typically performed at least as well as MORL-d. However, too many restarts, and the algorithm performed quite poorly.

Table 7: Total number of concept changes detected. Average for 30 runs. The correct number is 0. Anything more is a false positive.

Algorithm	Experiment Number					
	1	2	3	4	5	6
CDMORL						
CDMORL	198.9	166.6	184.9	123.7	175.8	123.0
CDMORL-d	181.3	16.7	61.4	35.5	28.9	37.8
CDMORL-w	242.5	113.3	165.4	143.6	182.2	154.6
CDMORL-d-w	156.0	8.4	52.0	21.6	27.6	24.4
Batch-based CDMORL						
b-CDMORL	12.2	28.4	35.4	23.8	43.0	24.6
b-CDMORL-d	1.7	3.9	4.1	2.8	3.8	1.6
b-CDMORL-w	14.7	39.0	40.8	22.9	32.1	24.7
b-CDMORL-d-w	3.5	2.4	8.0	1.5	4.4	4.6
Single Objective Drift-Aware RL						
SO_SC	250.8	116.3	200.6	104.3	172.2	136.7
b-SO_SC	57.0	31.9	40.0	23.6	34.1	25.3
SO_MC	219.8	148.4	184.1	110.3	166.9	142.9
b-SO_MC	59.3	34.7	35.3	31.0	36.4	24.8

Table 8 shows the results of statistical significance testing. The table indicates that using batches resulted in improvement that is statistically significant. The same holds true for using decay. Algorithms utilizing WoLF did not have a statistically significant difference in false positive detection. Interestingly, multiple objective algorithms detected a different number of changes than single objective algorithms, but

using multiple objectives to track drift in a single objective agent did not have a statistically significant difference in the number of false positives.

Table 8: Statistical significance of concept change count by agent category.

	1	2	3	4	5	6
MO Batches	1.456e-39	3.107e-07	5.486e-18	4.695e-11	1.746e-09	1.844e-13
MO Decay	8.359e-06	2.961e-34	3.569e-22	2.950e-28	2.260e-30	1.027e-26
MO WoLF	1.794e-01	8.593e-01	5.034e-01	7.734e-01	8.287e-01	2.432e-01
Drift: M or S	5.136e-01	1.711e-01	9.938e-01	8.948e-01	1.639e-01	8.458e-01
Drift-b: M or S	9.504e-01	4.503e-01	3.792e-01	9.901e-02	6.187e-01	6.800e-01
MO vs SO	7.511e-09	6.765e-13	2.265e-09	1.080e-10	9.504e-10	5.768e-10

5.1.6 Test for Final Concept Count in Stationary MERLIN. Table 9 contains the number of concepts each algorithm had, at the end of each run, on average. As described in Chapter III, algorithms delete concepts when they appear to be noise. Also, an agent could detect a change, but return to a previous concept before the new concept was declared permanent, thereby not creating a new concept. The correct final number of concepts was 1. It should be noted that all twelve algorithms can reach a nearly perfect performance as γ approaches 0, but this table is for the more challenging and insightful case of $\gamma = 0.9$.

The batch algorithms with decay consistently had fewer than four concepts. The other four algorithms with batches and the two basic algorithms with decay always had fewer than 10 concepts. The remaining algorithms often had 10-15 concepts.

Table 10 reports the statistical significance between types of algorithms. Two types of algorithms consistently had statistical significance in the final concept count, those using batches and those using decay. Algorithms that learn multiple objectives consistently achieved a better hypervolume than the single objective agents (which learn the hypervolume), and this result is significant. However, using multiple objectives to track drift did not make a significant impact.

Table 9: Final concept count. Average for 30 runs. The correct answer is 1.

Algorithm	Experiment Number					
	1	2	3	4	5	6
CDMORL						
CDMORL	9.48	8.55	11.93	8.41	12.21	9.66
CDMORL-d	7.90	3.48	7.03	3.97	3.93	4.21
CDMORL-w	11.10	9.38	11.90	8.48	11.90	9.03
CDMORL-d-w	9.59	3.24	6.59	4.59	4.90	5.17
Batch-based CDMORL						
b-CDMORL	4.14	6.52	8.07	4.97	7.14	6.41
b-CDMORL-d	2.03	2.69	2.90	2.41	2.72	2.17
b-CDMORL-w	4.48	6.66	8.45	4.79	7.55	5.28
b-CDMORL-d-w	2.34	2.24	3.62	2.00	3.00	2.66
Single-Objective Drift-Aware RL						
SO_SC	13.62	8.86	12.45	8.45	11.59	9.93
b-SO_SC	7.59	6.66	8.52	5.31	7.55	5.52
SO_MC	13.86	8.62	11.62	8.66	11.31	9.10
b-SO_MC	8.17	6.93	7.83	4.93	6.93	6.17

Table 10: Statistical significance of final concept counts by agent category.

	Experiment Number					
	1	2	3	4	5	6
MO Batches	5.330e-39	8.813e-05	3.720e-12	2.952e-14	1.350e-07	7.214e-12
MO Decay	2.762e-06	3.617e-33	1.203e-23	4.386e-22	8.463e-32	5.067e-24
MO WoLF	4.982e-02	9.859e-01	7.663e-01	8.935e-01	3.914e-01	9.929e-01
Drift: M or S	4.888e-01	6.395e-01	3.389e-01	6.377e-01	7.292e-01	1.339e-01
Drift-b: M or S	1.245e-01	4.060e-01	1.723e-01	3.146e-01	3.207e-01	2.302e-01
MO vs SO	1.701e-18	4.183e-12	1.507e-08	1.114e-09	2.526e-10	3.122e-09

5.2 Non-Stationary MERLIN Tests

The next set of experiments induces concept drift at intervals of 1000 steps. Each concept occurs four times. Actions rotate, but otherwise the environment is the same. That is to say, for a given node s_1 , the action that transitions to node s_2 changes its index, but the transition $s \rightarrow s'$ still exists, and the action a that causes that transition still has the same $R(s, a)$ function.

5.2.1 Accuracy Test: CDMORL Agents. Figures 7 - 12 show the plots of CDMORL without batches for the six non-stationary experiments. In these experiments, the performance of CDMORL-d and CDMORL-d-w are typically more accurate than the non-decay algorithms. The three non-decay algorithms, including the reference

MORL, perform similarly. MORL-d typically achieves the lowest (worst) accuracy. In several experiments, CDMORL-d and CDMORL-d-w appear to nearly converge to the optimal performance for all concepts. Each concept has the exact same optimum cycle of states possible, only the action sequence to achieve that optimum cycle changes with each concept. In the instantaneous hypervolume graph, CDMORL-d appears to find a good local optima several times. This is particularly evident in Figure 7 and Figure 11. Similar to the results seen in the stationary MERLIN, MORL-d tends to converge to a mediocre solution. When drift happens, it appears to stay stuck in that solution.

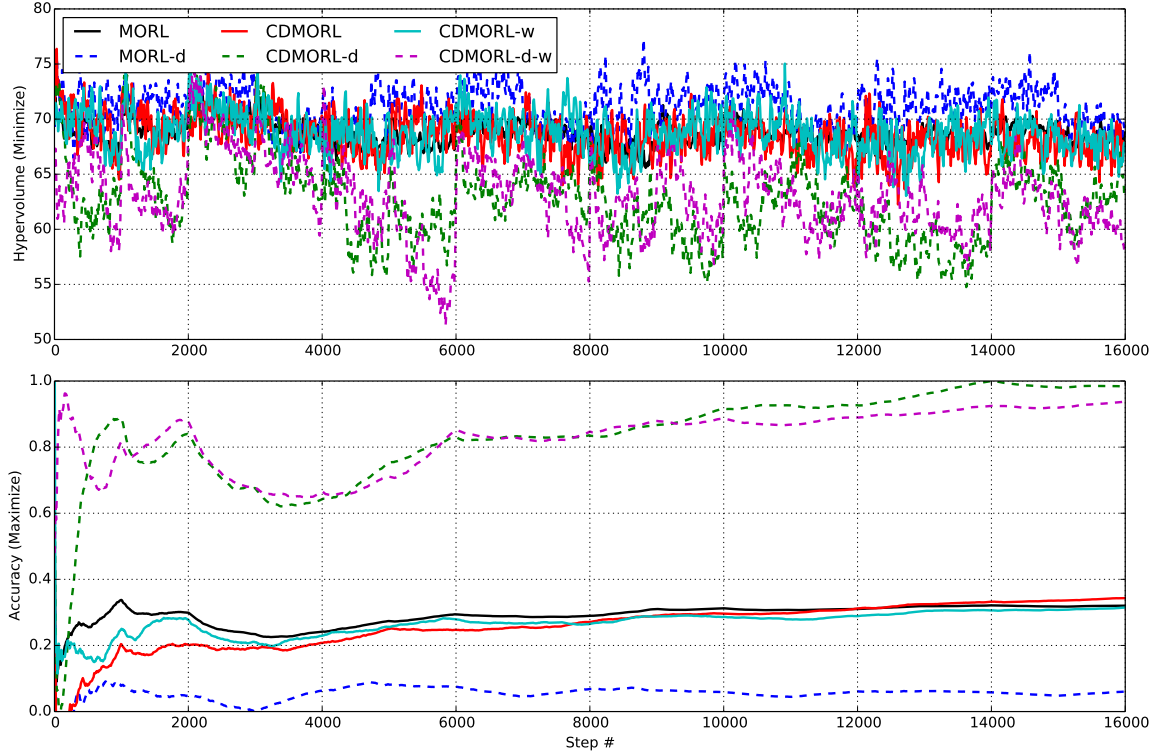


Figure 7: Hypervolume and accuracy for drifting MERLIN experiment 7, comparison among non-Batch agents and reference agents.

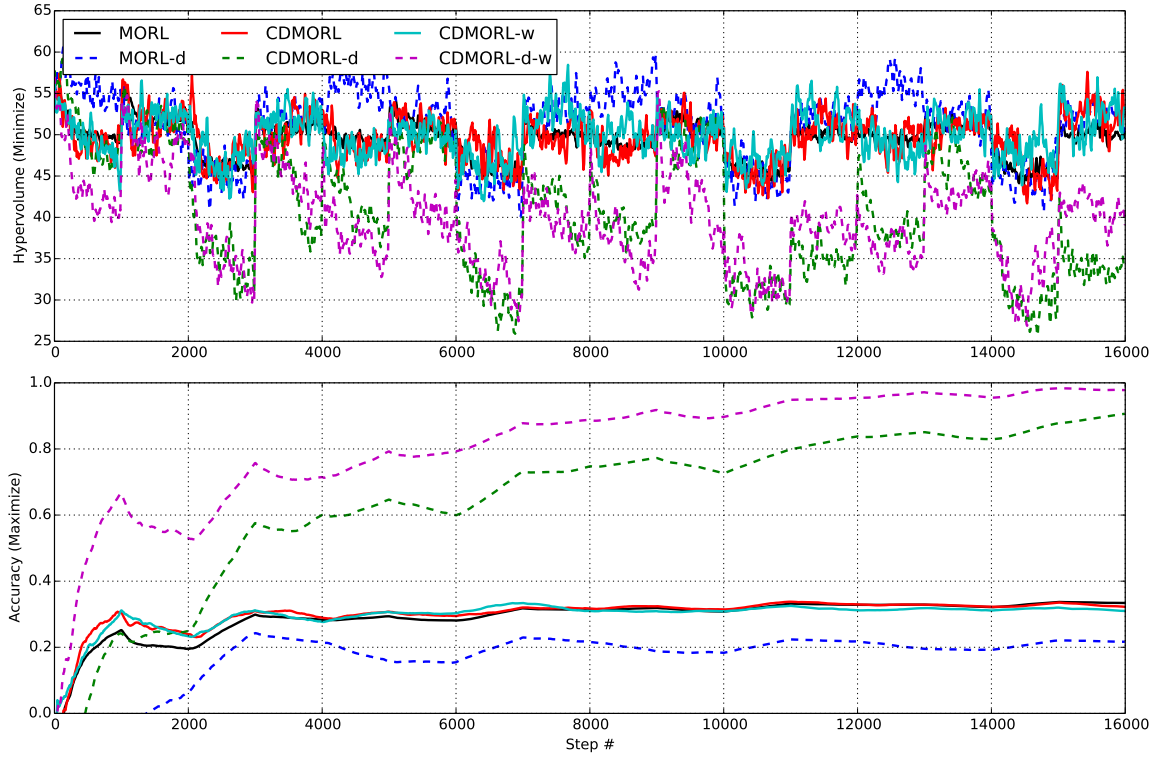


Figure 8: Hypervolume and accuracy for drifting MERLIN experiment 8, comparison among non-Batch agents and reference agents.

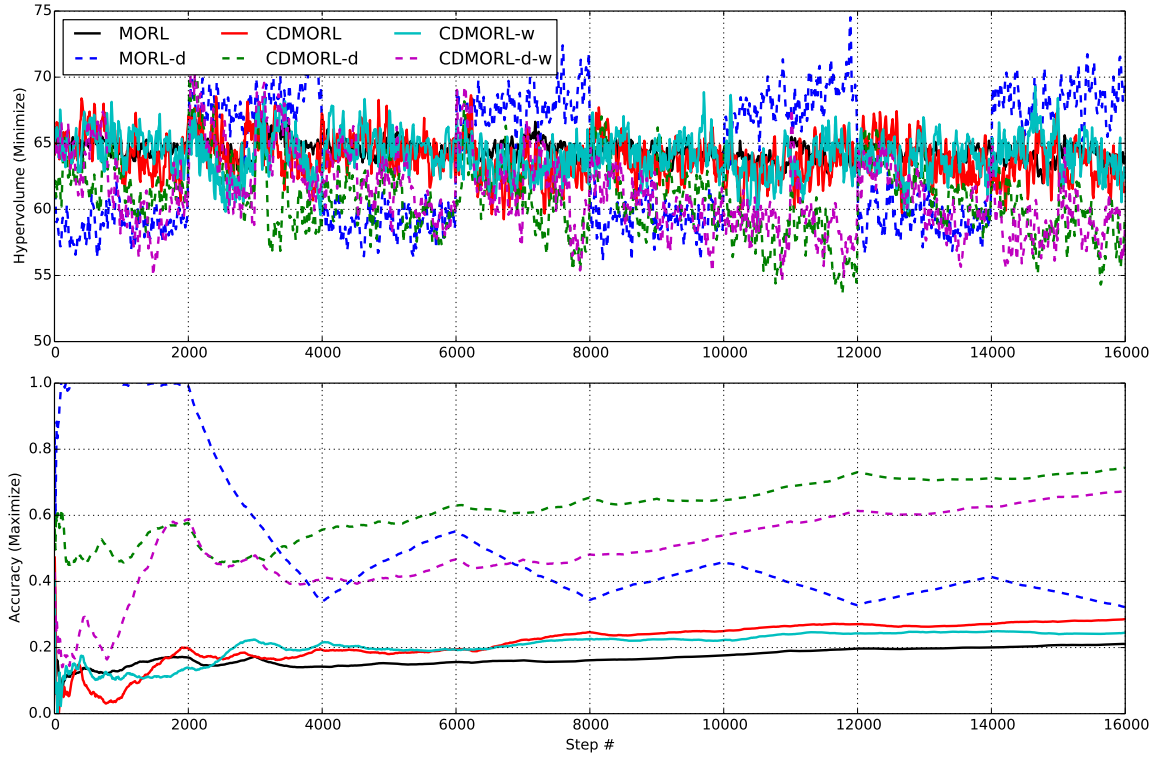


Figure 9: Hypervolume and accuracy for drifting MERLIN experiment 9, comparison among non-Batch agents and reference agents.

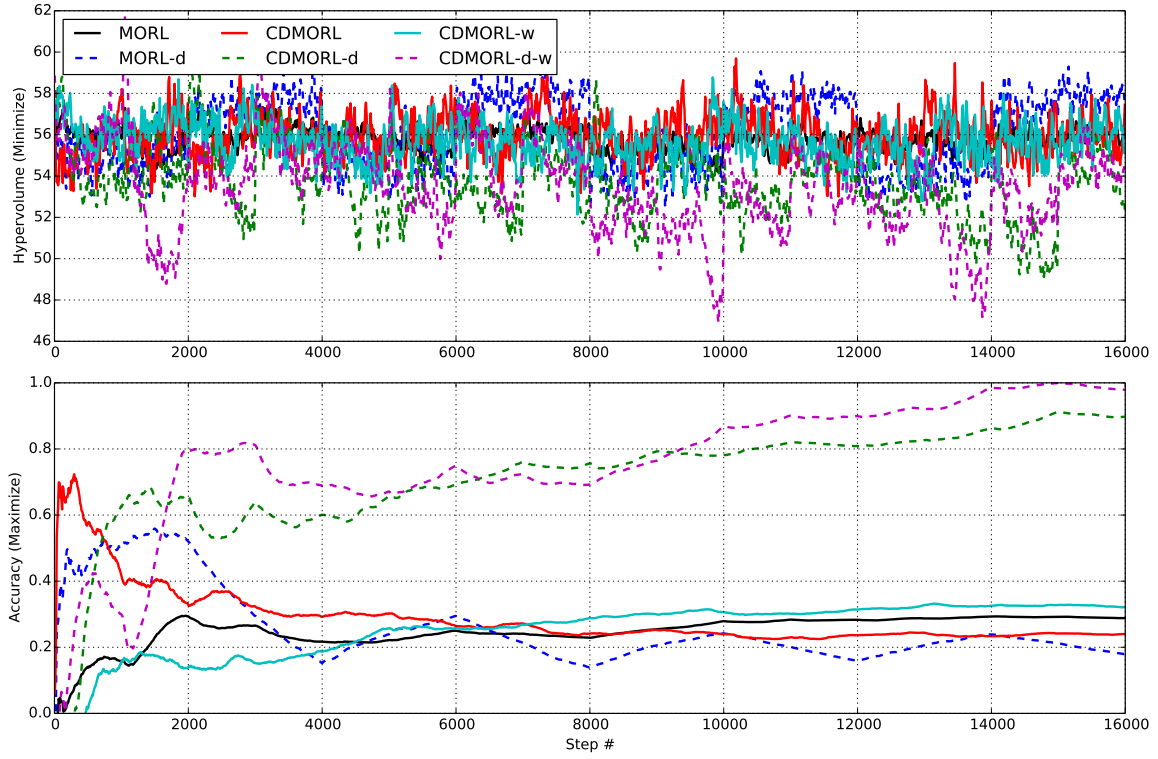


Figure 10: Hypervolume and accuracy for drifting MERLIN experiment 10, comparison among non-Batch agents and reference agents.

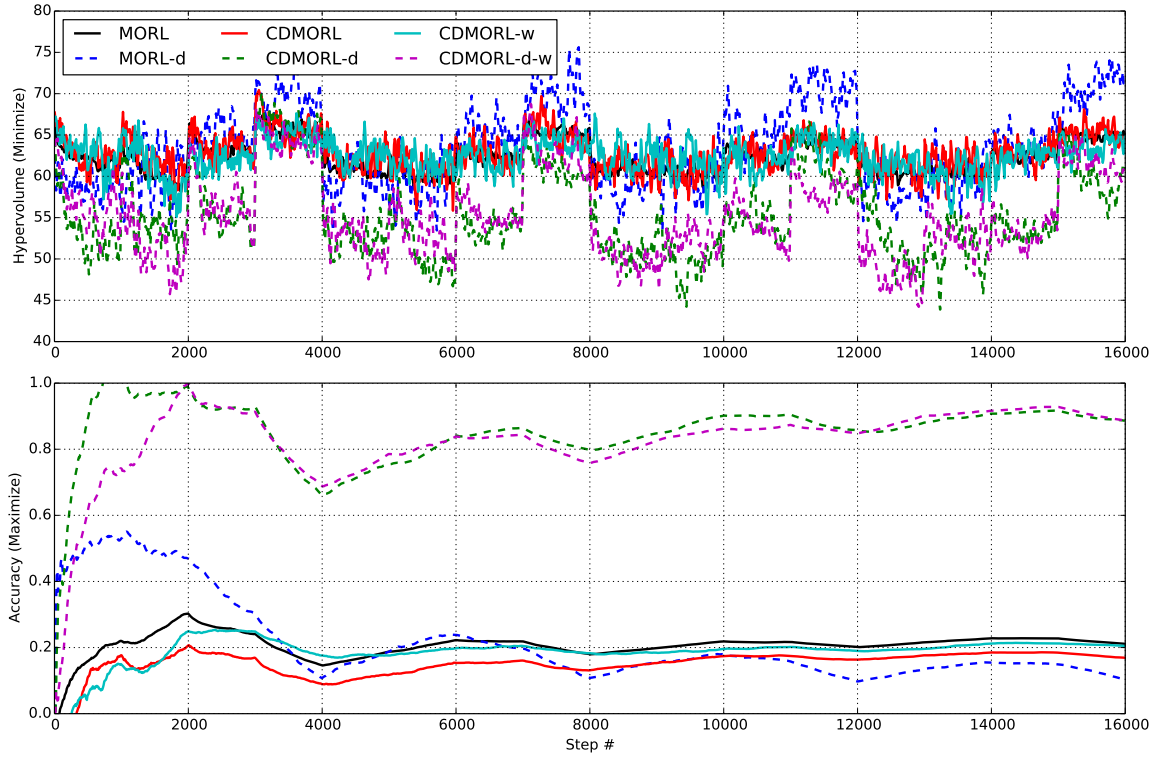


Figure 11: Hypervolume and accuracy for drifting MERLIN experiment 11, comparison among non-Batch agents and reference agents.

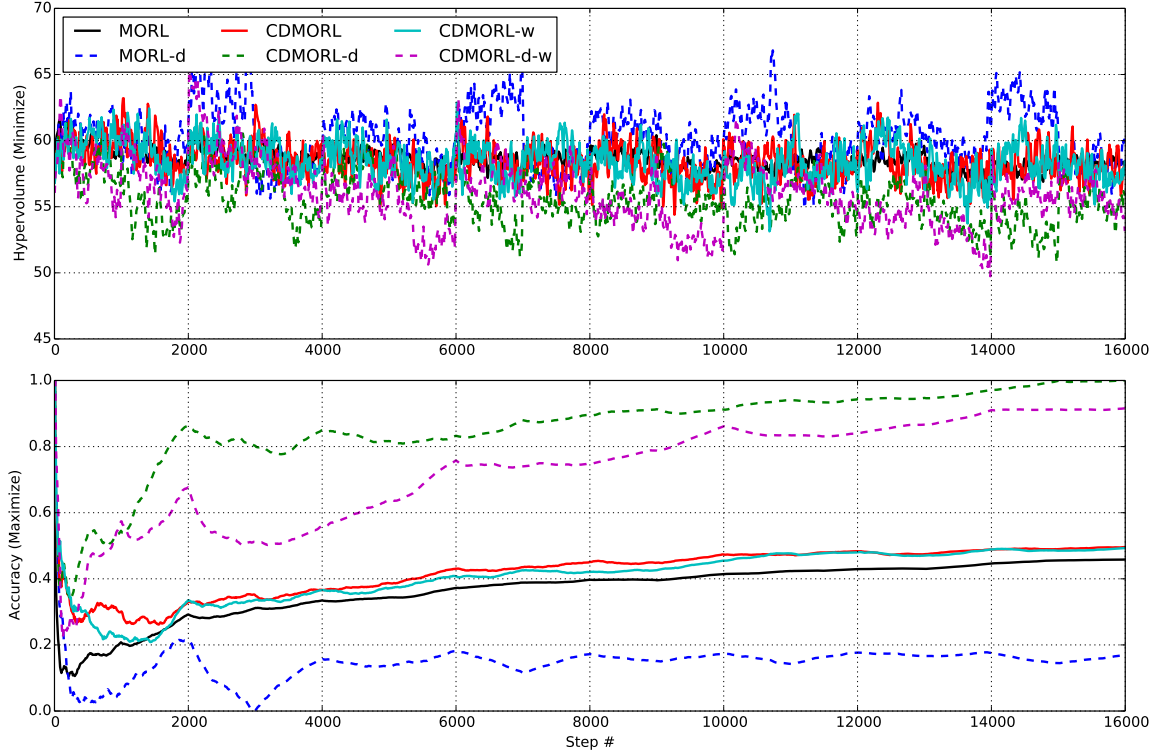


Figure 12: Hypervolume and accuracy for drifting MERLIN experiment 12, comparison among non-Batch agents and reference agents.

5.2.2 *Accuracy Test: Batch-based Agents.* Figures 13 - 18 are plots of the performance of CDMORL with batches in non-stationary MERLIN. As with the non-batch agents, the CDMORL agents with decay tend to perform noticeably better in both hypervolume and accuracy than the non-decay agents of all kinds, which all noticeably outperform MORL-d. The non-decay agents are grouped closely relative to the others, indicating little difference in performance in spite of any interventions present. The agents with decay, excluding MORL-d, seem to react to drift. The non-decay algorithms seem to maintain a consistently mediocre performance.

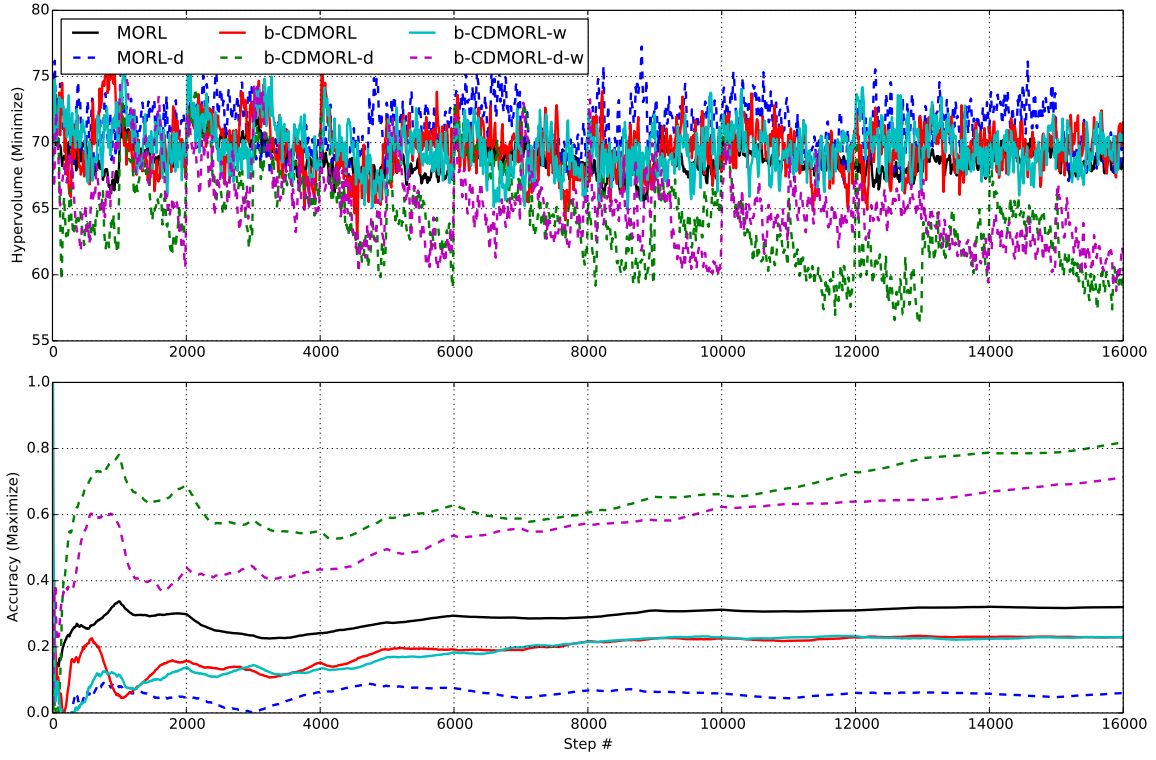


Figure 13: Hypervolume and accuracy for drifting MERLIN experiment 7, comparison among Batch agents and reference agents.

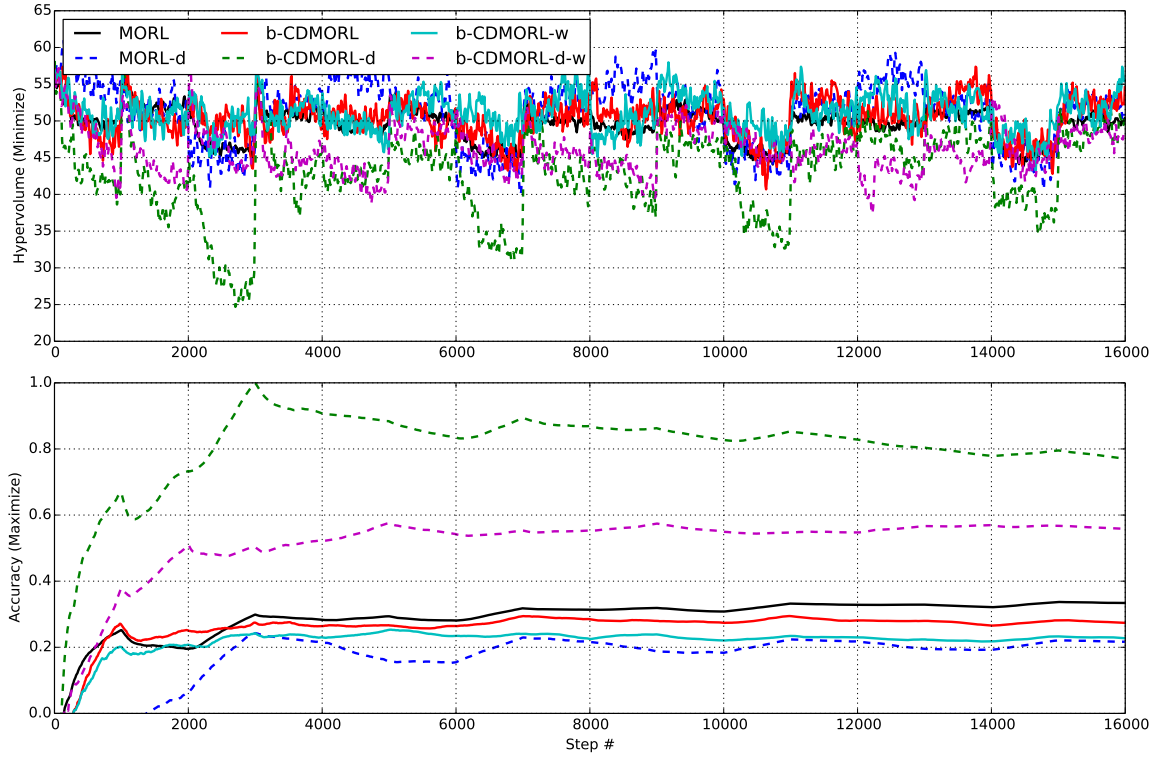


Figure 14: Hypervolume and accuracy for drifting MERLIN experiment 8, comparison among Batch agents and reference agents.

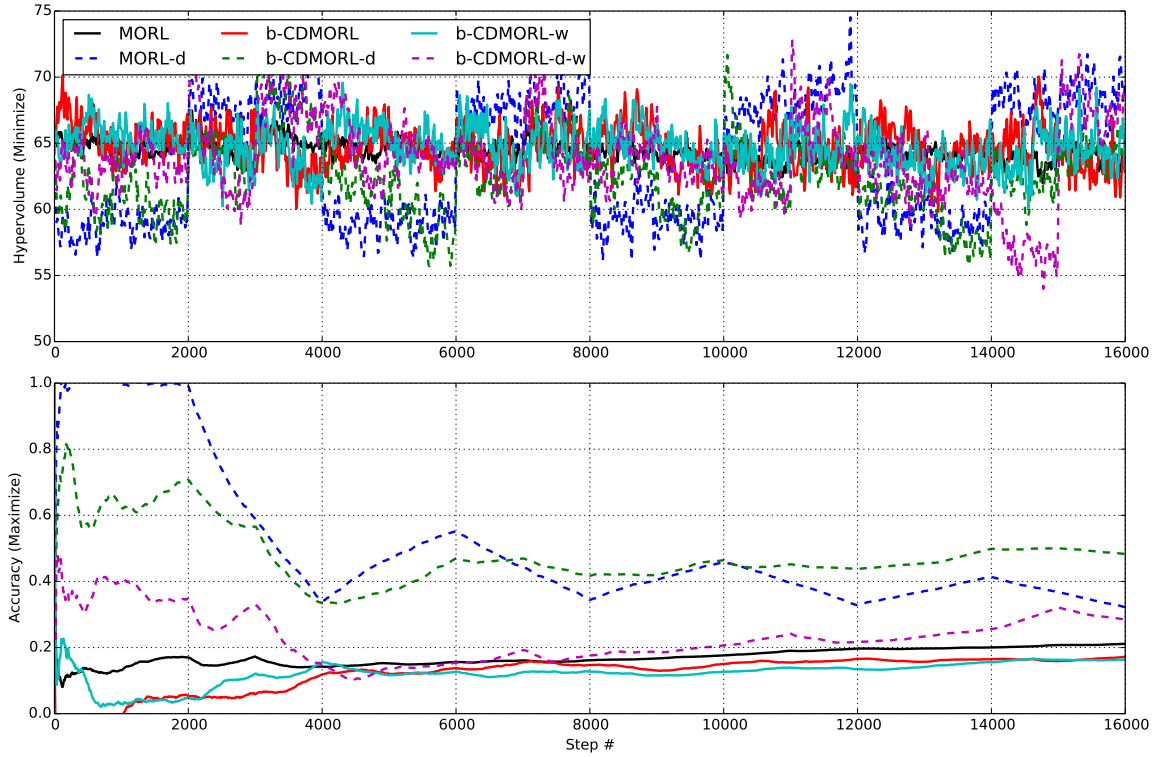


Figure 15: Hypervolume and accuracy for drifting MERLIN experiment 9, comparison among Batch agents and reference agents.

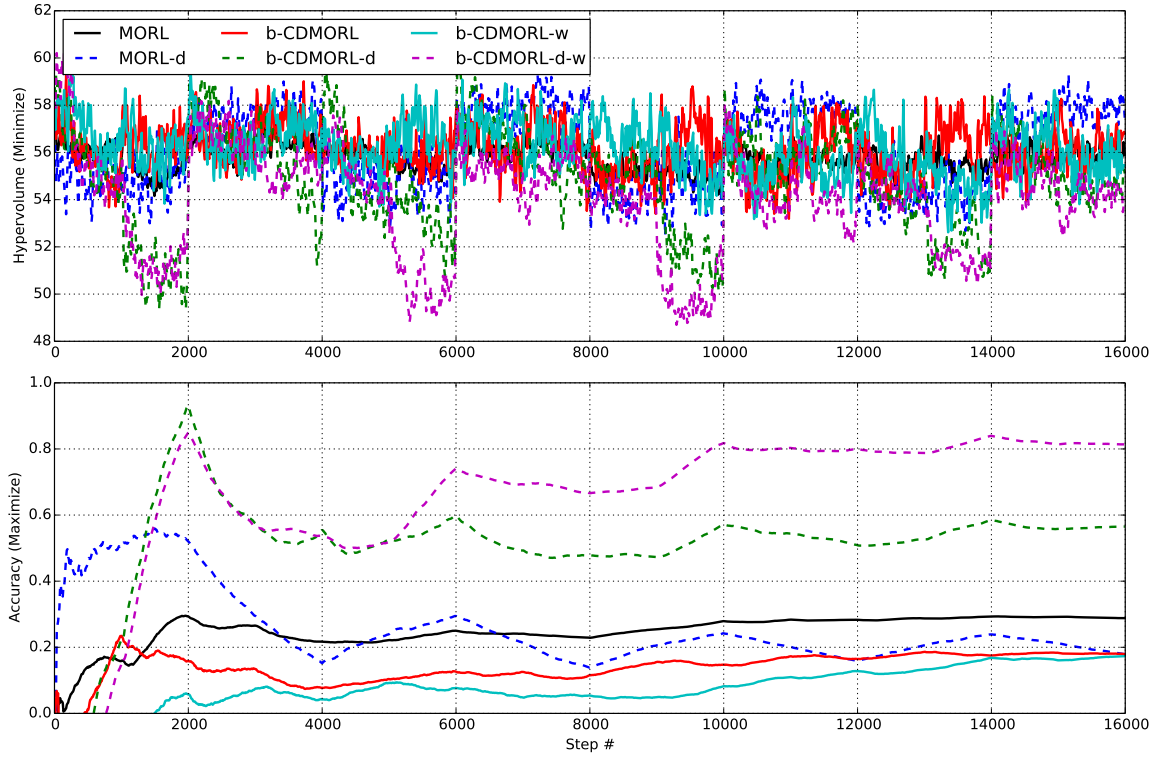


Figure 16: Hypervolume and accuracy for drifting MERLIN experiment 10, comparison among Batch agents and reference agents.

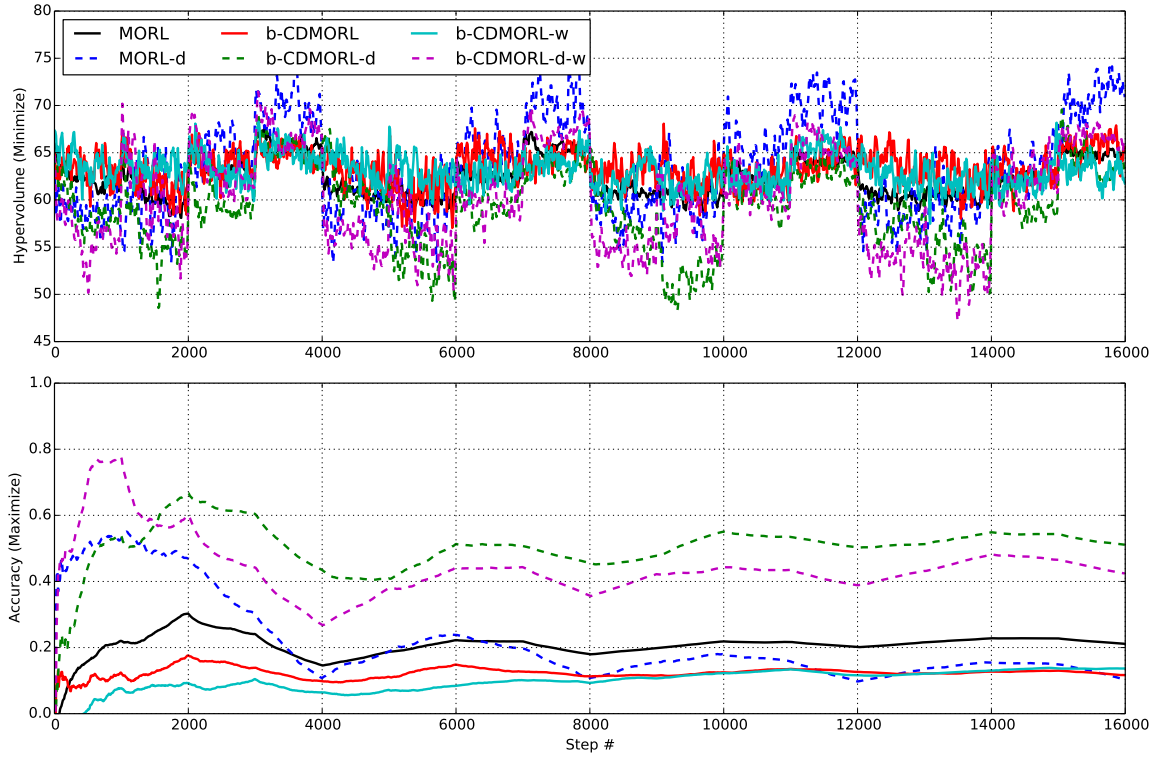


Figure 17: Hypervolume and accuracy for drifting MERLIN experiment 11, comparison among Batch agents and reference agents.

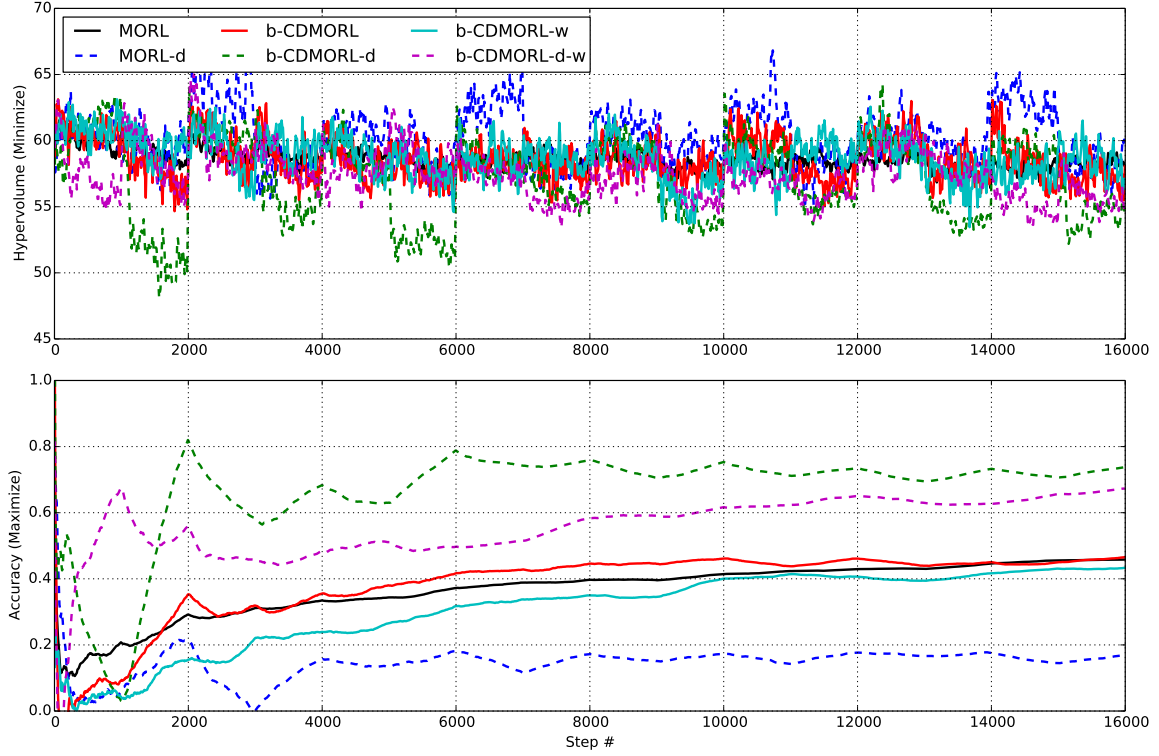


Figure 18: Hypervolume and accuracy for drifting MERLIN experiment 12, comparison among Batch agents and reference agents.

5.2.3 *Accuracy Test: Single Objective Agents.* Figures 19 - 24 are plots of the four drift-detecting single objective RL agents are shown, along with the two reference MORL agents and a reference SORL agent. The exact same smoothing process was used on these algorithms as the multi-objective algorithms, indicating a high degree of volatility in the single objective algorithms. The overall accuracy of all seven algorithms is poor. Of the drift-detecting SORL algorithms, those not using batches performed the best, presumably because this is a deterministic problem and the small advantage gained from learning immediately rather than N_B steps later made a difference. The performance of those algorithms is minimally better than plain MORL and MORL-d.

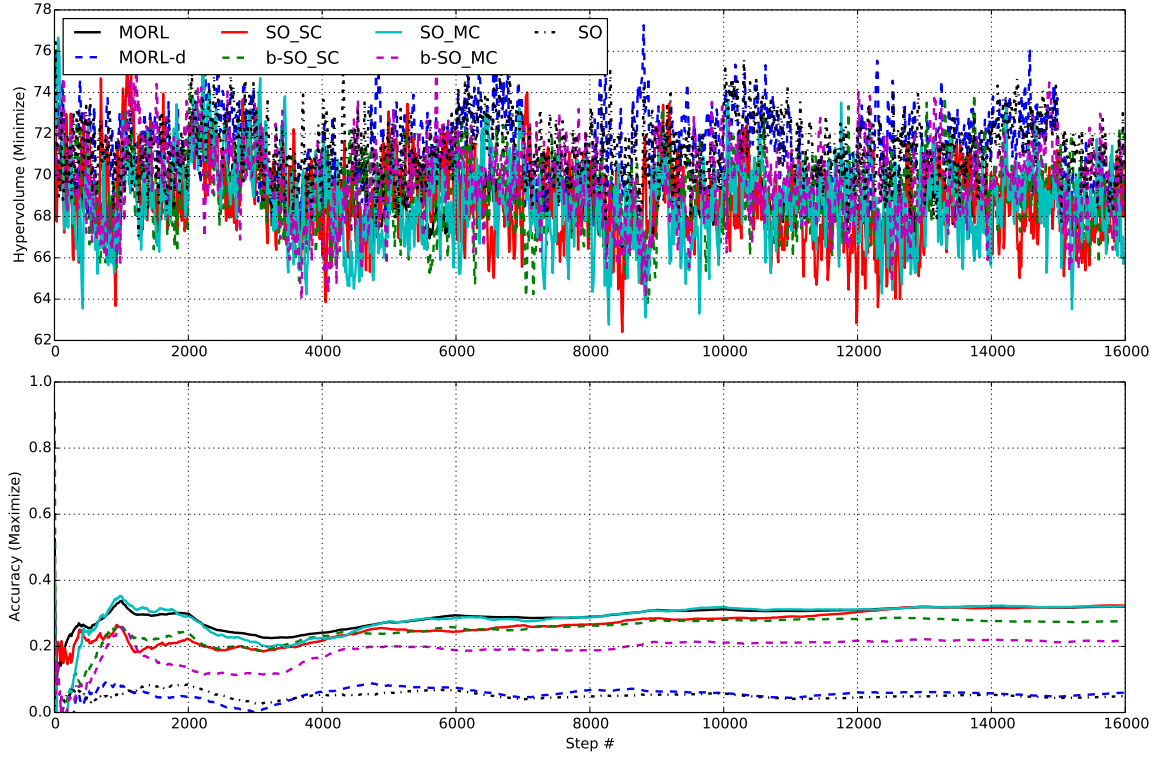


Figure 19: Hypervolume and accuracy for drifting MERLIN experiment 7, comparison among single objective agents and reference agents.

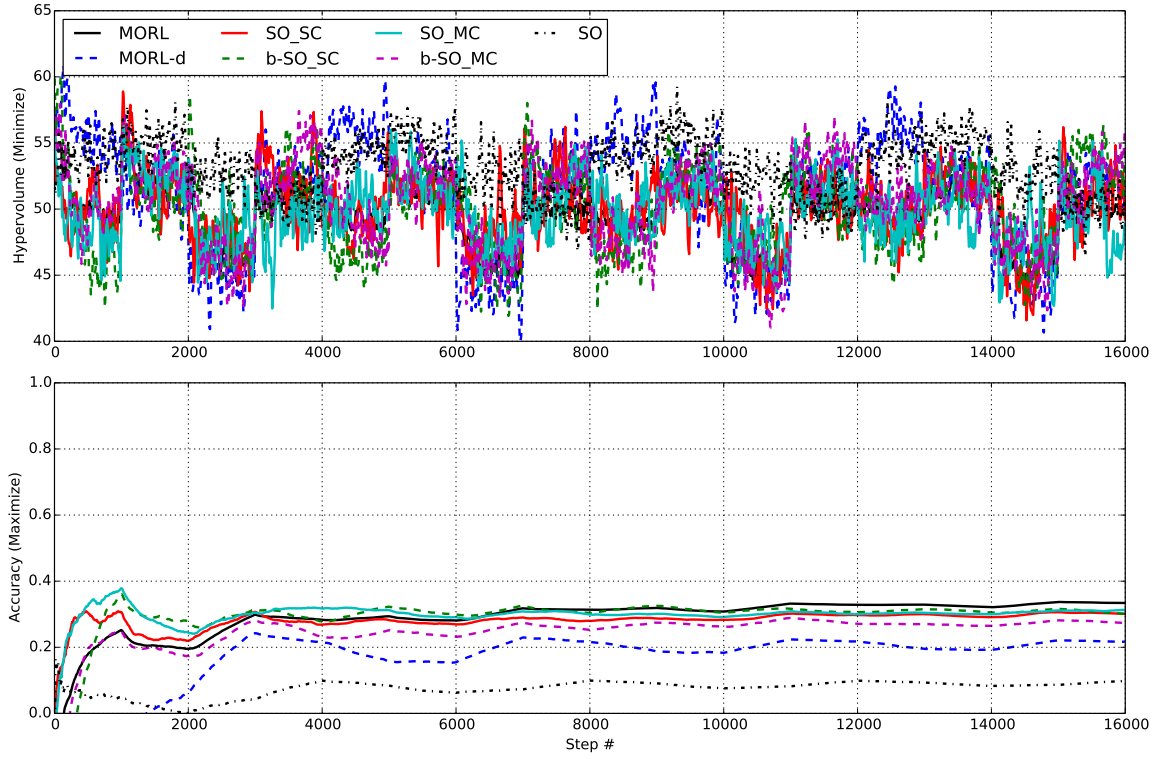


Figure 20: Hypervolume and accuracy for drifting MERLIN experiment 8, comparison among single objective agents and reference agents.

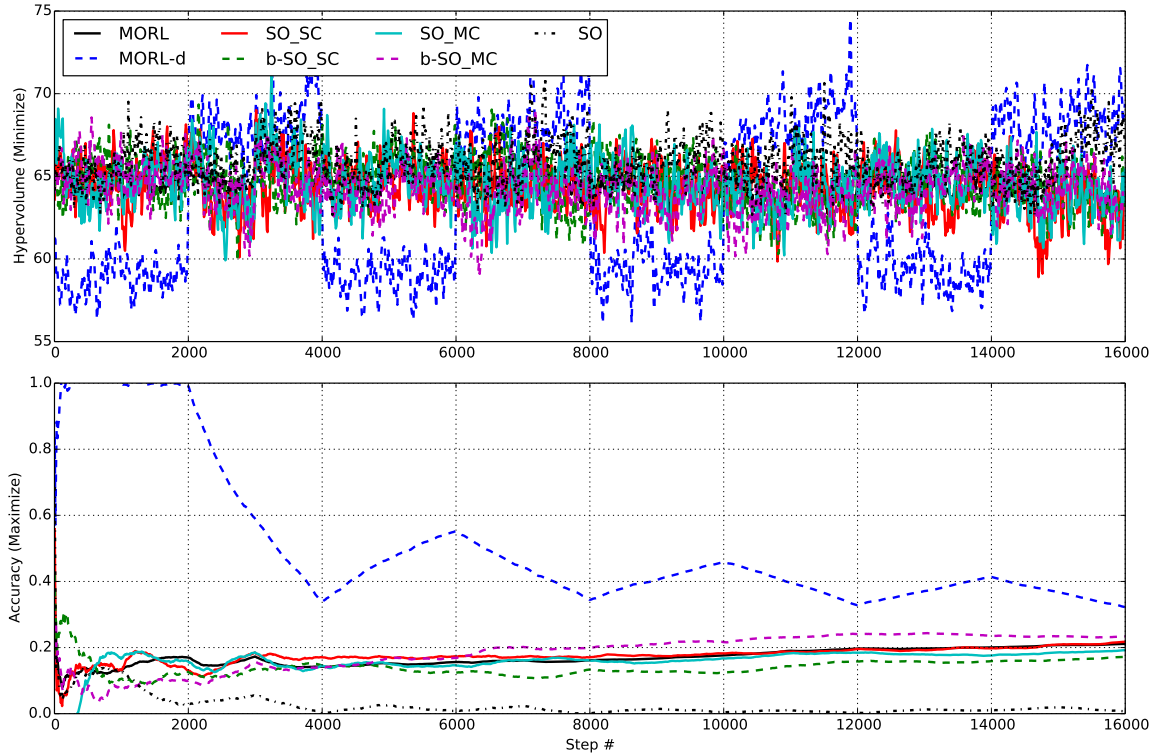


Figure 21: Hypervolume and accuracy for drifting MERLIN experiment 9, comparison among single objective agents and reference agents.

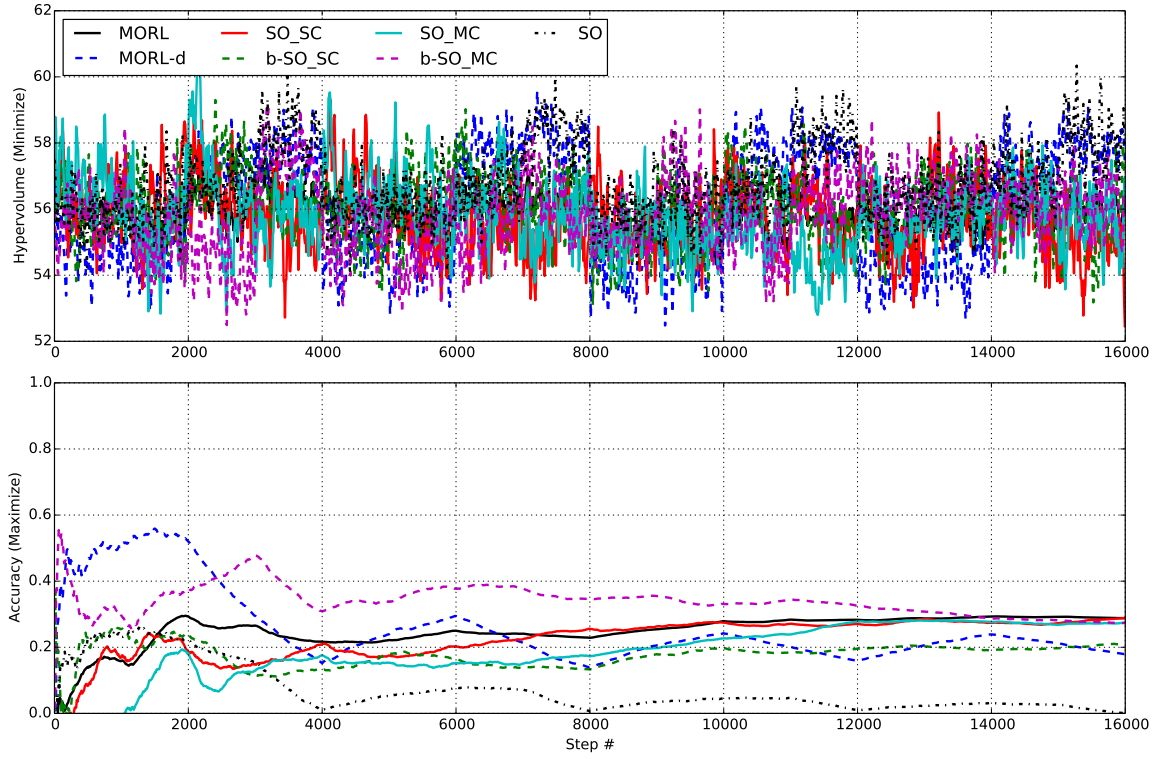


Figure 22: Hypervolume and accuracy for drifting MERLIN experiment 10, comparison among single objective agents and reference agents.

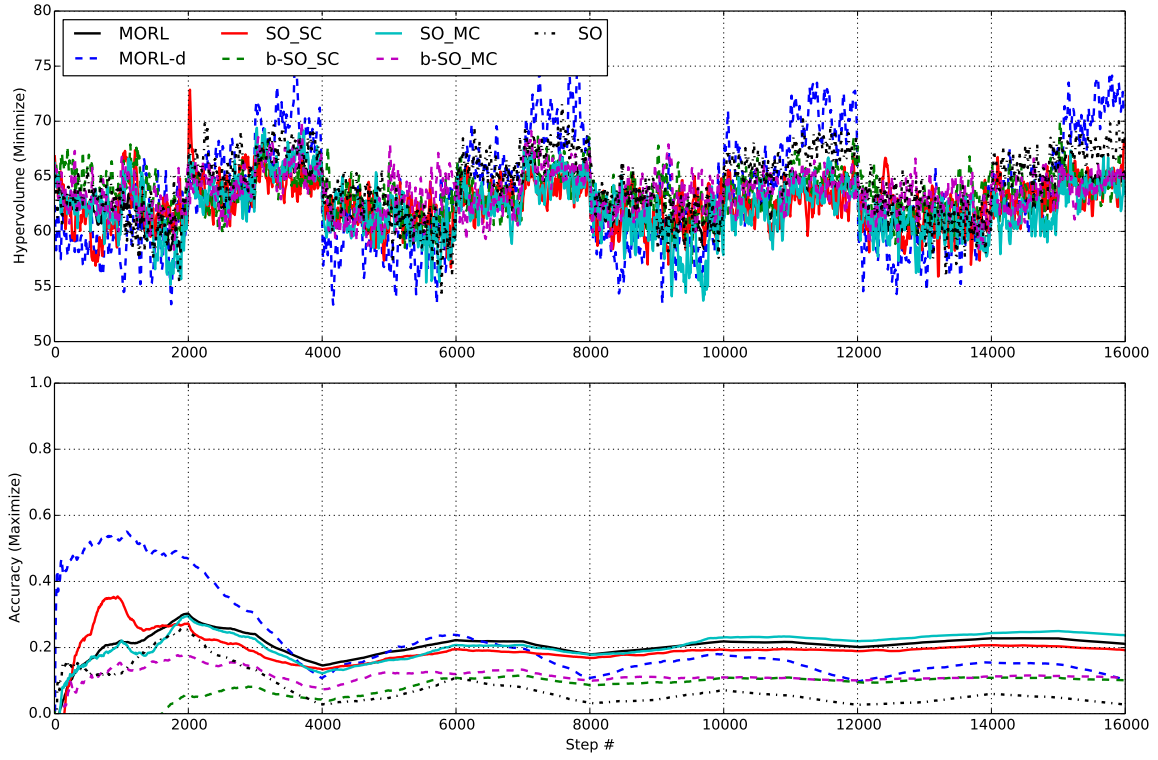


Figure 23: Hypervolume and accuracy for drifting MERLIN experiment 11, comparison among single objective agents and reference agents.

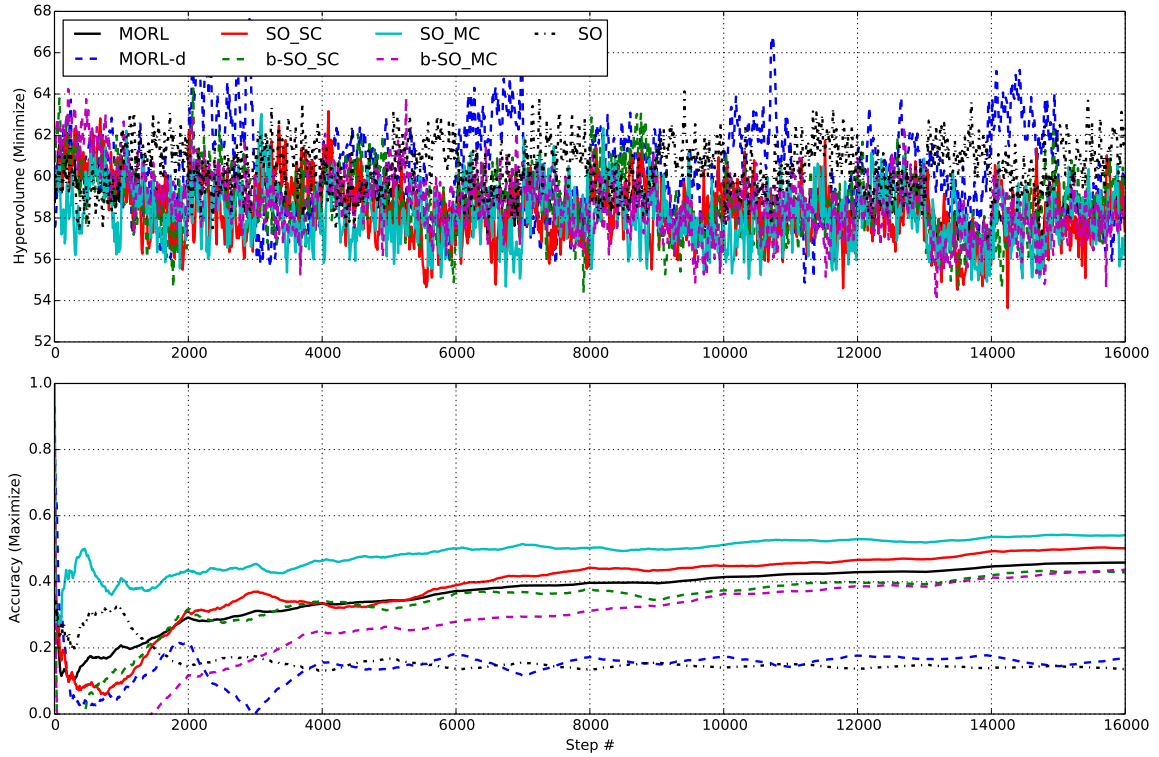


Figure 24: Hypervolume and accuracy for drifting MERLIN experiment 12, comparison among single objective agents and reference agents.

5.2.4 Summary of Accuracy Tests in Non-Stationary MERLIN. Table 11

presents the accuracy for the non-stationary experiments. Overall, the CDMORL algorithms without batches consistently performed the best, with decay as the greatest indicator of success.

Table 11: Summary of ending accuracy for the six non-stationary experiments. The best performer for each experiment within a category is **bold faced**, with the best overall *italicized*. For Experiment 11, the two bolded performances of the algorithms are within rounding error and deemed a tie.

Algorithm	Experiment Number					
	7	8	9	10	11	12
CDMORL						
CDMORL	0.343	0.322	0.286	0.240	0.169	0.496
CDMORL-d	<i>0.983</i>	0.907	<i>0.744</i>	0.898	<i>0.886</i>	<i>1.000</i>
CDMORL-w	0.315	0.310	0.245	0.321	0.205	0.493
CDMORL-d-w	0.938	<i>0.978</i>	0.674	<i>0.979</i>	<i>0.887</i>	0.916
Batch-based CDMORL						
b-CDMORL	0.229	0.274	0.172	0.179	0.117	0.466
b-CDMORL-d	0.818	0.770	0.483	0.565	0.511	0.738
b-CDMORL-w	0.229	0.227	0.163	0.173	0.137	0.433
b-CDMORL-d-w	0.713	0.558	0.284	0.814	0.424	0.674
Single-Objective Drift Aware RL						
SO_SC	0.324	0.301	0.218	0.289	0.192	0.502
b-SO_SC	0.275	0.301	0.172	0.208	0.101	0.429
SO_MC	0.322	0.313	0.192	0.274	0.237	0.541
b-SO_MC	0.216	0.273	0.234	0.274	0.109	0.436
Reference						
MORL	0.320	0.334	0.211	0.288	0.211	0.458
MORL-d	0.060	0.217	0.322	0.179	0.103	0.168
SO	0.049	0.098	0.007	0.000	0.028	0.136

5.2.5 Test for Concept Change Count in Non-stationary MERLIN. Table 12

shows the number of concept changes detected by the various *CDMORL* algorithms. For this set of experiments, there should be exactly 15 changes detected. Generally, false positives were very high, except for the four batch methods.

Table 12: Total number of concept changes that were detected. Average for 30 runs. The correct number is 15. Anything more is guaranteed to be a false positive.

Algorithm	Experiment Number					
	7	8	9	10	11	12
CDMORL						
CDMORL	1087.8	502.9	731.5	606.1	776.3	522.2
CDMORL-d	441.0	71.6	291.7	204.3	151.7	158.8
CDMORL-w	1206.2	571.3	713.0	562.3	718.3	540.0
CDMORL-d-w	459.2	87.9	293.2	180.0	167.9	163.8
Batch-based CDMORL						
b-CDMORL	309.1	140.7	160.6	125.7	171.1	92.0
b-CDMORL-d	63.4	36.4	48.3	38.1	44.4	28.4
b-CDMORL-w	285.8	140.9	166.0	133.4	158.3	97.4
b-CDMORL-d-w	62.8	37.9	50.7	29.1	40.0	29.4
Single Objective Drift-Aware RL						
SO_SC	1164.0	541.7	709.5	588.7	701.7	523.6
b-SO_SC	285.4	133.6	160.0	121.6	161.9	92.5
SO_MC	1110.3	585.4	697.7	579.7	645.7	522.8
b-SO_MC	297.2	137.2	164.5	116.0	165.4	91.7

Table 13 reports the statistical significance in the false positives. Agents using batches, decay, or multiple objectives were different with significant results.

Table 13: Statistical significance of concept change count by agent category.

	Experiment Number					
	7	8	9	10	11	12
MO Batches	7.225e-23	5.706e-11	3.782e-28	2.588e-17	3.364e-15	1.446e-21
MO Decay	2.246e-20	6.347e-35	1.376e-16	5.502e-24	6.685e-30	5.550e-22
MO WoLF	9.774e-01	8.342e-01	7.684e-01	8.633e-01	8.818e-01	6.812e-01
Drift: M or S	3.158e-01	4.554e-01	6.297e-01	7.148e-01	2.867e-01	7.736e-01
Drift-b: M or S	6.917e-01	9.938e-01	9.504e-01	7.975e-01	3.427e-01	6.802e-01
MO vs SO	1.508e-07	8.569e-13	3.253e-06	4.336e-08	3.690e-10	7.762e-08

5.2.6 *Test for Final Concept Count in Non-stationary MERLIN.* In Table 14 is the average final number of permanent concepts each algorithm retained. The correct answer is 4. All algorithms discovered too many concepts, with the batch-based algorithms with drift finding the fewest.

Table 14: Final concept count. Average for 30 runs. The correct answer is 4.

Algorithm	Experiment Number					
	7	8	9	10	11	12
CDMORL						
CDMORL	53.62	38.10	47.00	33.86	43.79	36.34
CDMORL-d	31.07	18.10	28.03	20.21	23.14	21.31
CDMORL-w	54.21	36.83	47.31	34.79	44.69	36.62
CDMORL-d-w	32.45	18.21	30.34	21.41	22.24	20.55
Batch-based CDMORL						
b-CDMORL	31.00	28.10	32.38	19.21	30.83	21.17
b-CDMORL-d	16.34	15.21	16.97	11.90	15.59	12.86
b-CDMORL-w	30.62	27.90	29.86	19.31	29.31	21.21
b-CDMORL-d-w	15.66	13.41	18.28	11.41	14.83	12.03
Single-Objective Drift-Aware RL						
SO_SC	52.79	37.76	49.38	33.10	44.34	36.41
b-SO_SC	31.28	27.07	30.41	19.41	29.41	21.31
SO_MC	56.66	38.03	48.59	35.52	44.59	33.21
b-SO_MC	30.66	27.41	31.83	19.34	30.48	21.41

Table 15 shows statistical significance results for final concept count. The multi-objective agents performed better than the single objective agents of any type. The result is significant with batches and decay, though which had a more significant result alternates between the two. In experiment 12, there was a significant result for the single objective non-batch agent detecting drift with multiple objectives in contrast to the agent detecting drift with a single objective. This appears to be an anomaly, especially considering that neither algorithm consistently outperformed the other.

CDMORL with decay in its four forms achieved the overall greatest accuracy. This can be seen in the instantaneous hypervolume when these agent consistently achieved lower values than the others and quickly re-achieved good performance after concept change. The batch-based algorithms with decay did not perform as well as the non-batch algorithms with decay in terms of accuracy in the non-stationary experiments, which is surprising because batch-based algorithms had the fewest false

Table 15: Statistical significance of final concept counts by agent category.

	Experiment Number					
	7	8	9	10	11	12
MO Batches	1.909e-25	3.972e-07	3.318e-19	6.203e-26	5.410e-11	1.117e-21
MO Decay	1.550e-20	6.677e-38	2.471e-25	2.329e-18	2.201e-35	1.096e-21
MO WoLF	8.864e-01	5.420e-01	8.555e-01	6.687e-01	5.427e-01	7.682e-01
Drift: M or S	3.761e-02	7.611e-01	7.373e-01	5.777e-02	9.813e-01	8.330e-03
Drift-b: M or S	5.524e-01	4.969e-01	2.960e-01	5.307e-01	1.868e-01	9.687e-01
MO vs SO	2.061e-08	4.597e-13	2.807e-11	5.988e-07	1.011e-12	6.265e-08

positives and the most precise concept count. Algorithms with decay achieved fewer false positives and more precise concept counts than those without decay. Accordingly, the batch-based algorithms with decay achieved the fewest false positives and the most precise concept count. However, these did not perform as well as their non-batch counterparts.

Overall, algorithms using WoLF are negligibly different from those not using WoLF. From examining the instantaneous hypervolume, the WoLF agents tended to be more conservative, typically not achieving hypervolume values that were more extreme than their non-WoLF counterparts. The lack of benefit could be because MERLIN is deterministic, or it could be that WoLF isn’t necessary given the intervention of mitigating catastrophic forgetting.

5.3 Resource Gathering

Experiments #13-#18 cover the resource gathering (RG) domain tests. The RG domain simulates tactical decision making [3]. The RG domain tests the ability of CDMORL agents to mitigate drift when indications of drift are infrequent. Concepts last 2500 steps.

5.3.1 Accuracy Test: CDMORL Agents. Figures 25-30 show plots of the instantaneous hypervolume and the accuracy for the four CDMORL agents not using batches as well as the two reference MORL implementations. There is one figure for each of the six different instantiations of the resource gathering task.

Experiment 13 is shown in Figure 25. This shows that concepts 1 and 4 are fairly similar, but concepts 2 and 3 are much more challenging. This is illustrated by the large variance for all agents, and is probably symptomatic of the random chance to be killed by an enemy that is close to the home node or by enemies close to treasure. The three algorithms using decay all markedly outperform those not using decay.

Figure 26 has a much lower variance in its four concepts. Only *MORL-d* has a significant problem with the second concept, likely indicative that the path it took to treasure in the first concept takes it through enemies for the second concept, and it simply never learns to perform differently. The results of this one differs from Experiment 1 in that *MORL-d* performs so much worse than the others, going from third to last.

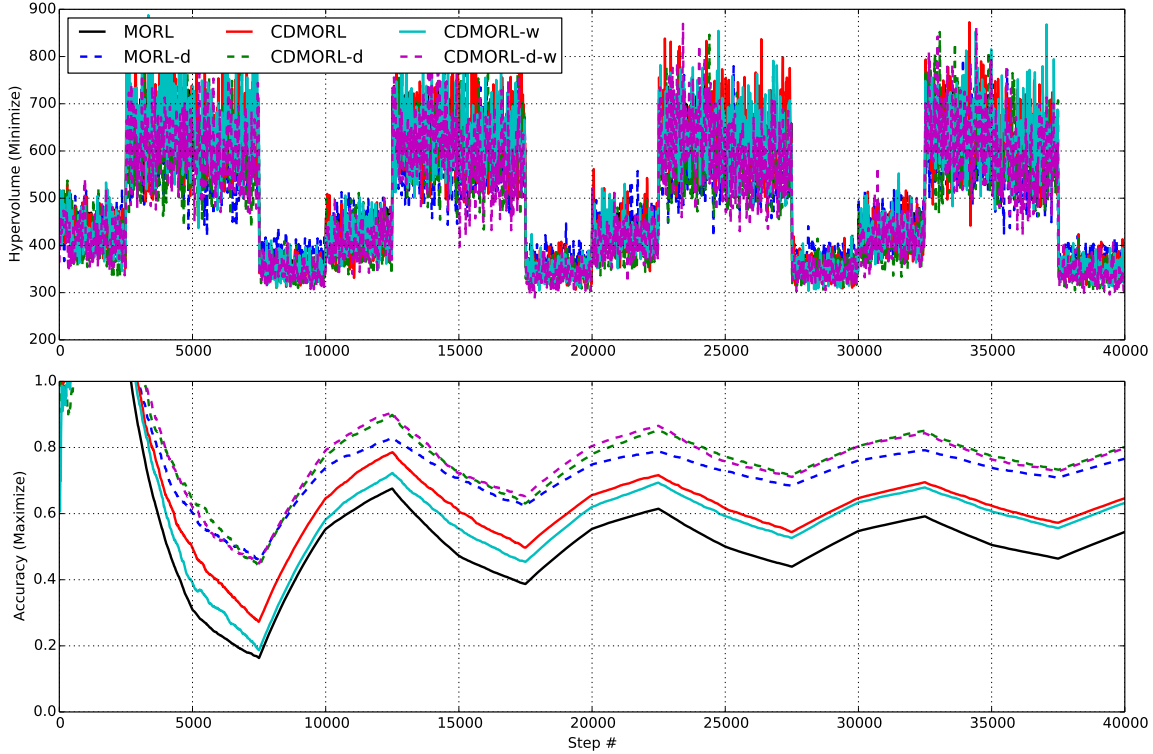


Figure 25: Hypervolume and accuracy for resource gathering experiment 13, comparison among non-Batch agents and reference agents.

Clear performance bands are visible in Figure 27. *CDMORL-d* and *CDMORL-d-w* consistently performing the best on instantaneous hypervolume. With the exception of *MORL-d* trading off with *CDMORL*, no other algorithm has an accuracy that rises

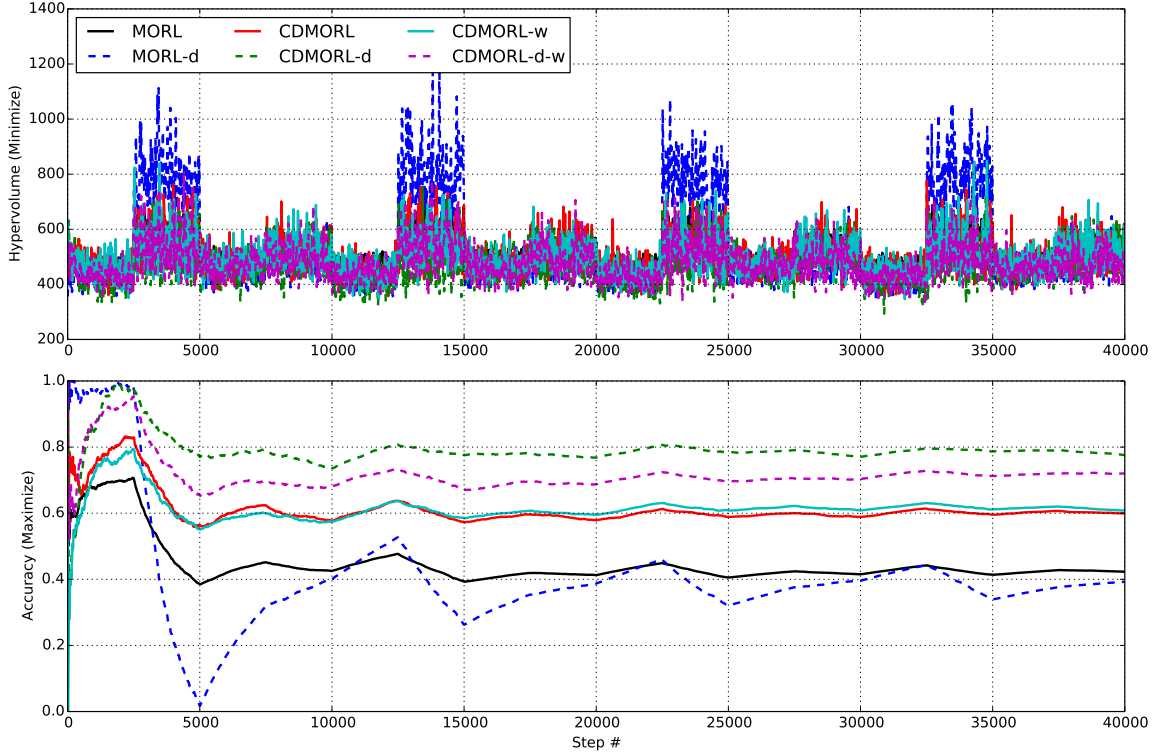


Figure 26: Hypervolume and accuracy for resource gathering experiment 14, comparison among non-Batch agents and reference agents.

above or drops below the accuracy of another algorithm. The algorithms detecting drift stay more accurate than the reference algorithms.

Figure 28 shows one easy concept followed by three hard concepts. This is inferred from the instantaneous hypervolume, which shows all agents achieving about the same, relatively low (good), value. However, in subsequent concepts, instantaneous hypervolume gets much worse for most agents. This is likely due to the location of the enemies with respect to the treasure in the graph. The decaying algorithms detecting drift are able to react each time.

Experiment 17, shown in Figure 29, has a marginally more difficult concept 2. All agents except for *MORL-d* are able to adjust reasonably quickly for the second concept.

Figure 30 shows a harder third and fourth concept. This figure too has visible performance bands in the instantaneous hypervolume. The impact of decaying but

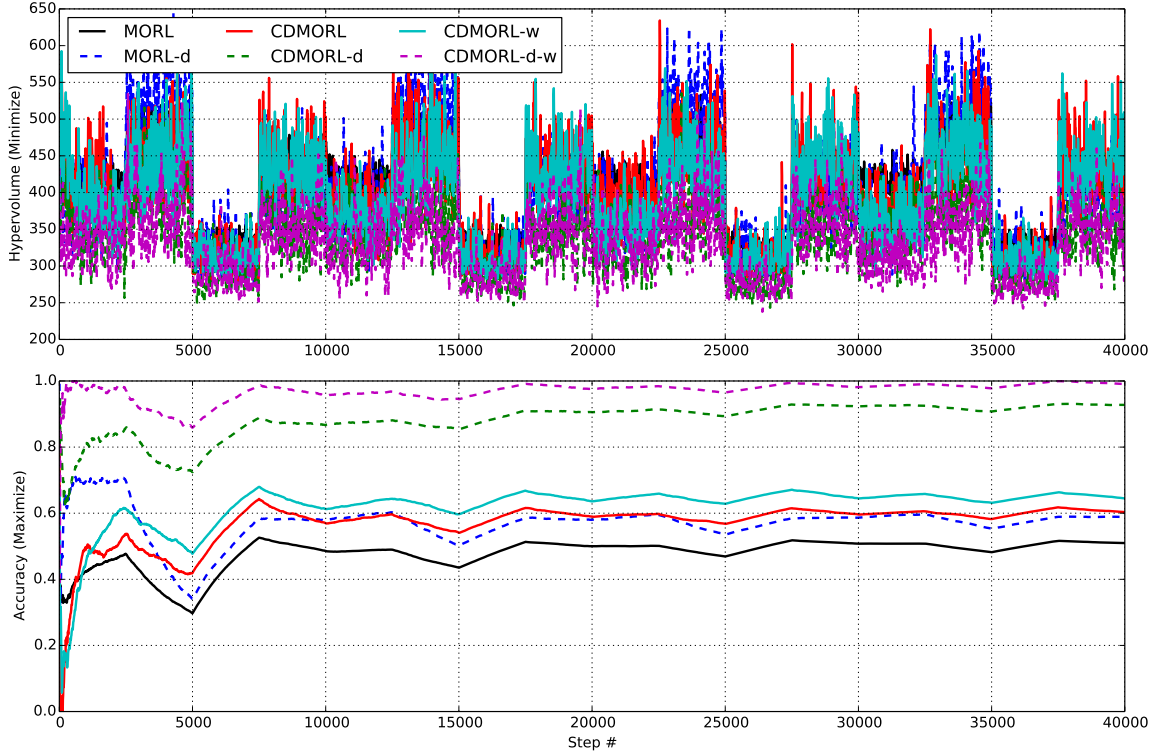


Figure 27: Hypervolume and accuracy for resource gathering experiment 15, comparison among non-Batch agents and reference agents.

not adjusting for concept change can be seen in the accuracy of *MORL-d*, which took a big turn for the worse when the third concept showed up, and it never recovered.

Throughout, it is interesting that the overall performance of the algorithms doesn't improve substantially. This could be indicative that local optima are learned very quickly and adhered to, or that the overlapping information between the many concepts prohibits additional improvement.

Overall, the best two agents were *CDMORL-d* and *CDMORL-d-w*. The next two best were usually *CDMORL* and *CDMORL-w*, which traded places. For these six agents, the relative performance was usually set after the first concept cycle, except for *MORL-d* which continued to decline for lack of plasticity.

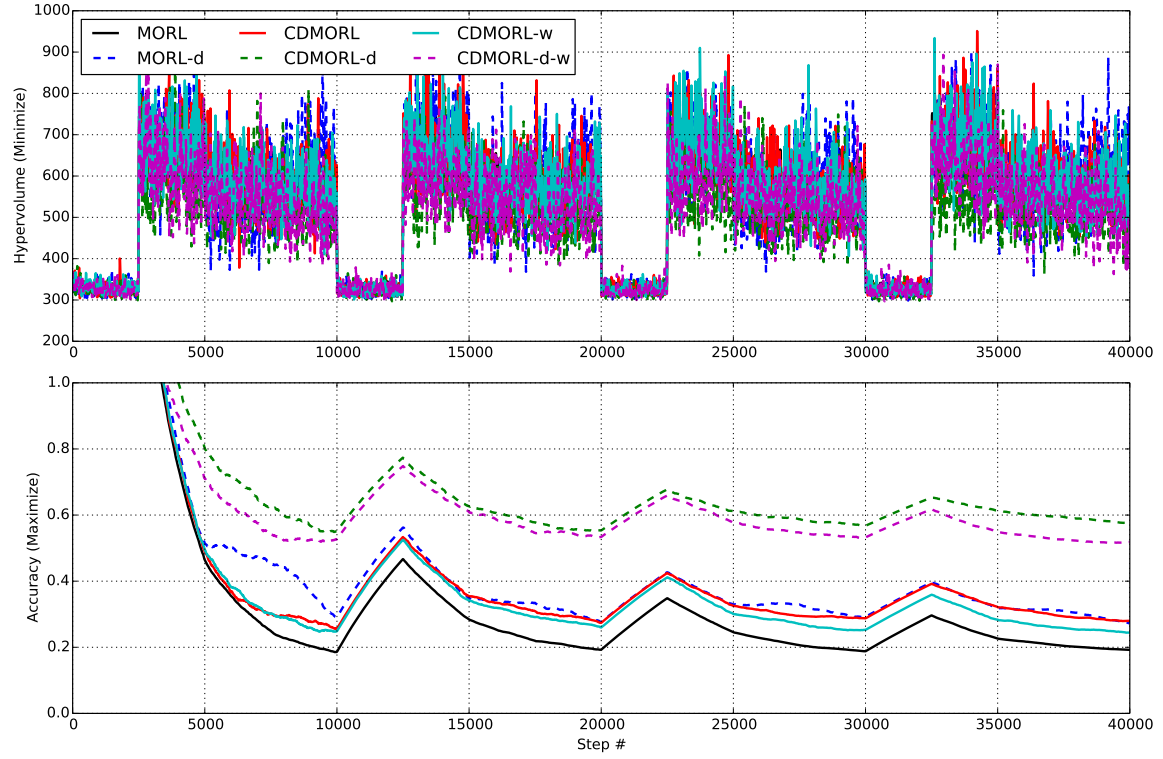


Figure 28: Hypervolume and accuracy for resource gathering experiment 16, comparison among non-Batch agents and reference agents.

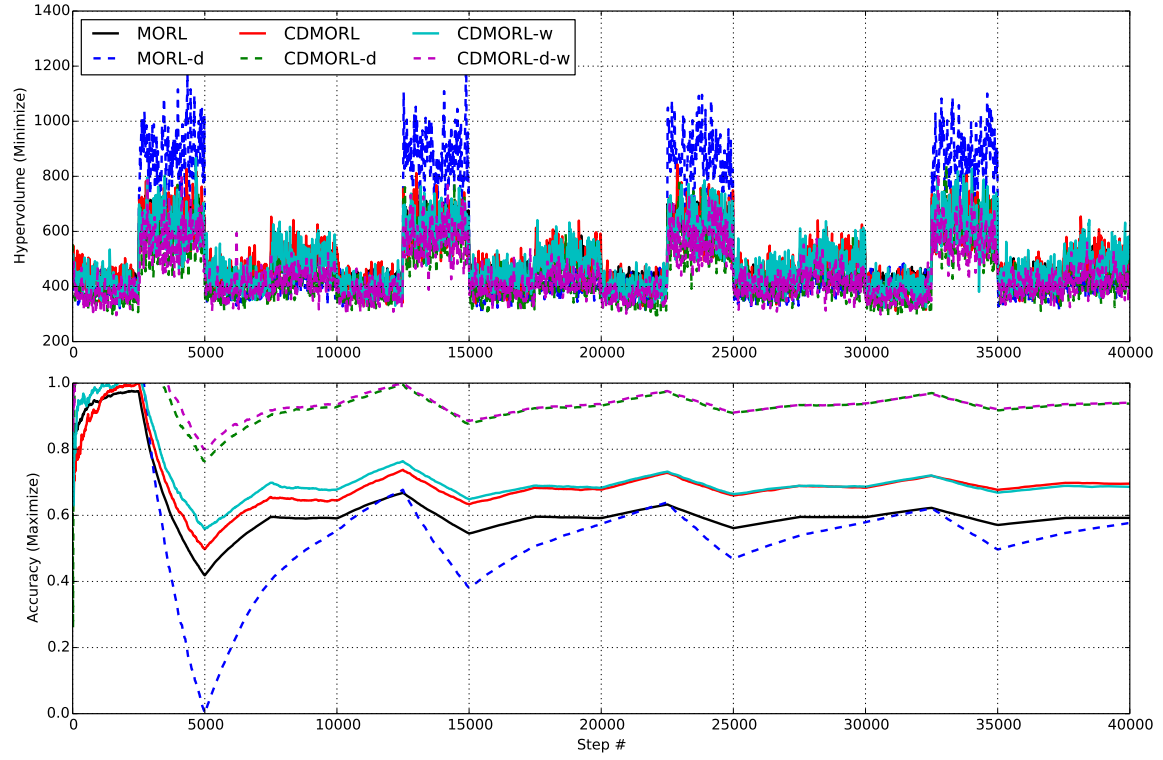


Figure 29: Hypervolume and accuracy for resource gathering experiment 17, comparison among non-Batch agents and reference agents.

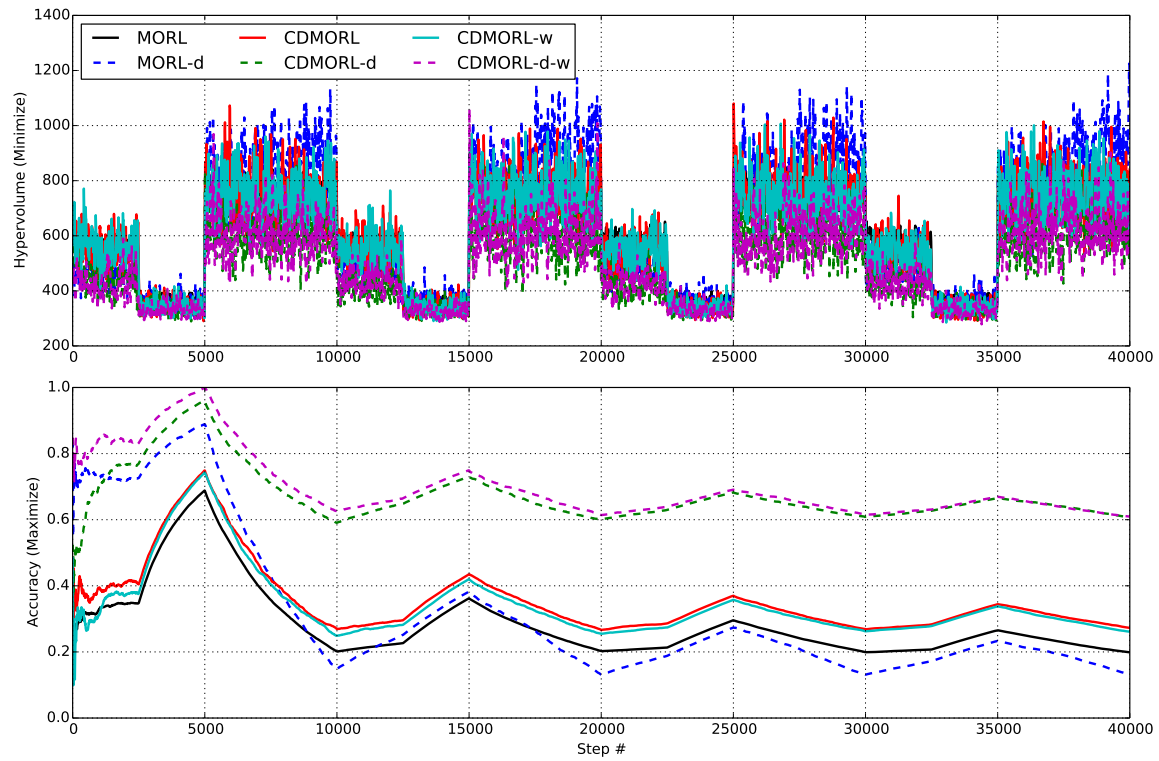


Figure 30: Hypervolume and accuracy for resource gathering experiment 18, comparison among non-Batch agents and reference agents.

5.3.2 *Accuracy Test: Batch-based Agents.* Figures 31-36 show Experiments 1-6 again, this time with batch-based CDMORL algorithms and the two reference algorithms.

Experiment 13, shown in Figure 31, has two phenomena not seen in the non-batch algorithms: algorithms frequently move up or down in relative performance to others, and the algorithms continue to improve over time. However, only *b-CDMORL-d* ends up outperforming the reference *MORL-d*. All algorithms have an upward performance trend, especially *b-CDMORL-d*.

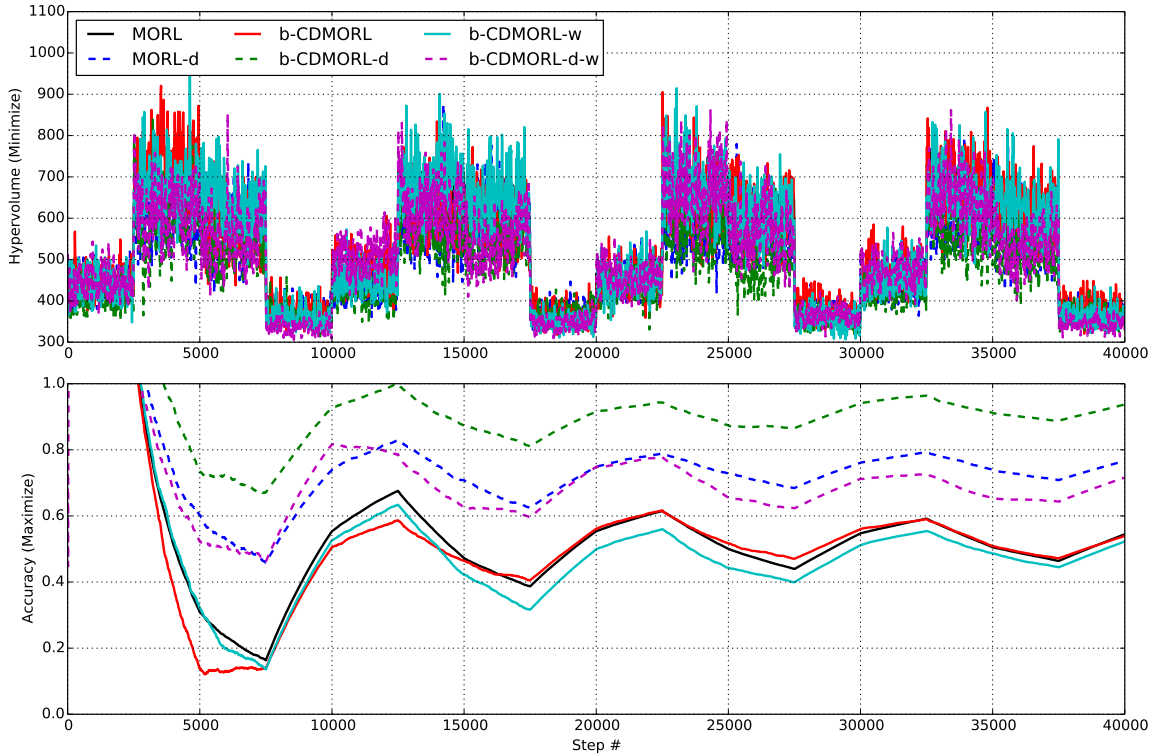


Figure 31: Hypervolume and accuracy for resource gathering experiment 13, comparison among Batch agents and reference agents.

Figure 32 (Experiment 14) shows that the relative performance for all algorithms except for *MORL-d* reach steady state by the end of the second cycle. Algorithms *b-CDMORL* and *b-CDMORL-w* both perform decisively worse than both reference algorithms.

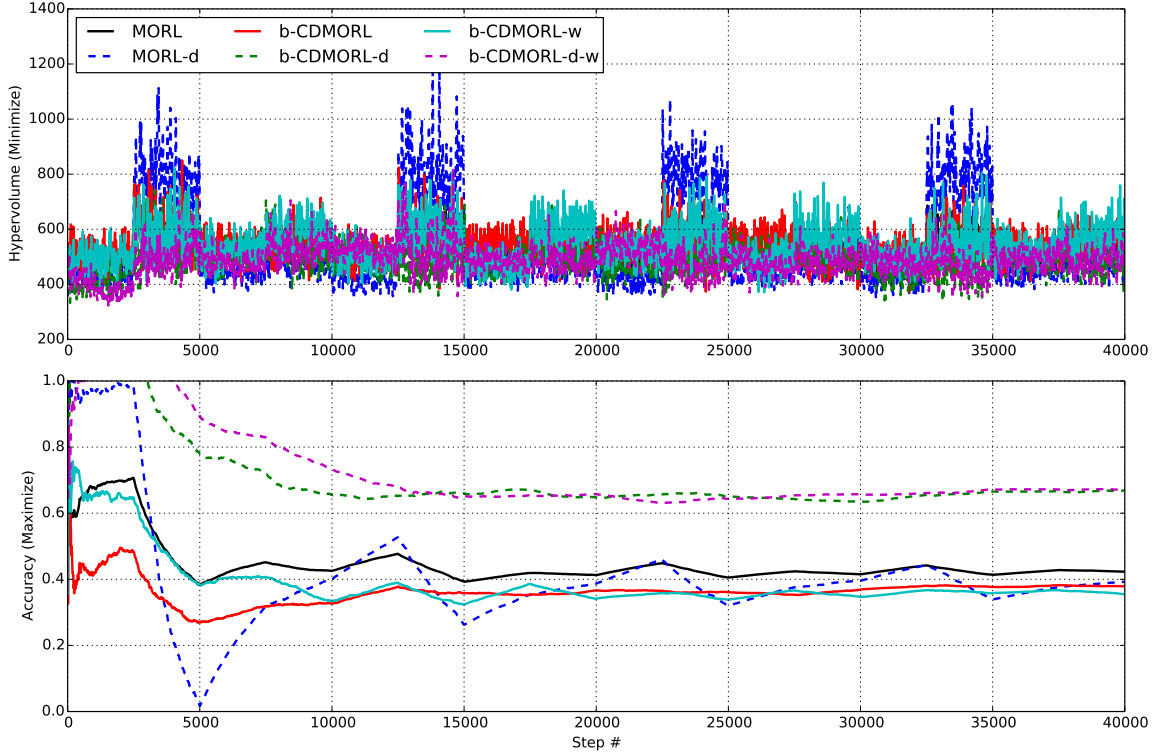


Figure 32: Hypervolume and accuracy for resource gathering experiment 14, comparison among Batch agents and reference agents.

Results from experiment 15 are plotted in Figure 33. Here, only *b-CDMORL-d-w* ends up outperforming *MORL-d*, and even then only narrowly. In this experiment, it took three cycles for the algorithms to reach steady state in the accuracy graph.

Figure 34 also reaches steady state in the accuracy chart around the third cycle. The *b-CDMORL* algorithm has a particularly hard time with the second concept, showing up most profoundly in, interestingly, the fourth cycle. While this could have been due to excessive thrashing in trivial concept recognitions in essentially identical concepts, this is ruled out by Table 17, which shows about 50 concept changes (which is two orders of magnitude less inaccurate than algorithms that markedly outperformed this one). It could be due to failure to classify a different concept or failure to recognize the correct concept. The exact cause is however unknown.

Figure 35 plots the results of experiment 17. It shows how WoLF reduces the volatility of instantaneous hypervolume across concept changes. Both *b-CDMORL-w* and *b-CDMORL-d-w* have the smallest drop in accuracy during the second concept in

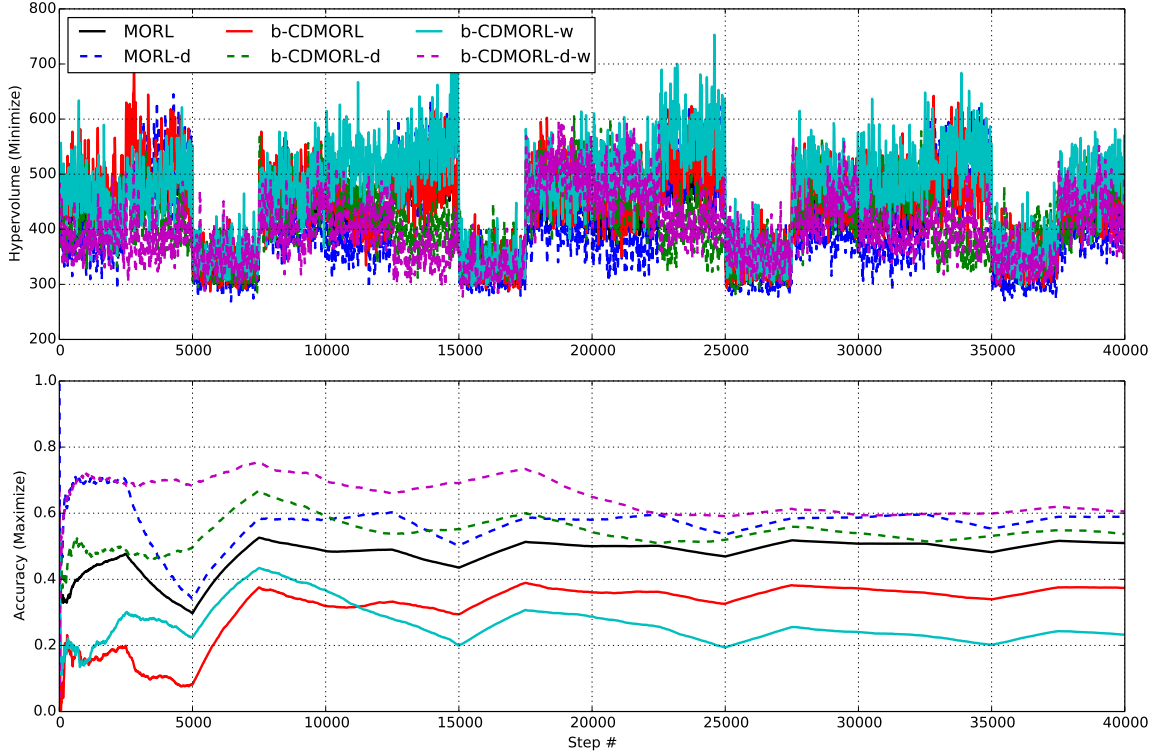


Figure 33: Hypervolume and accuracy for resource gathering experiment 15, comparison among Batch agents and reference agents.

the first cycle, a trend that continues, albeit with less impact, in the second cycle. It takes three cycles for the accuracies to reach steady state or an otherwise predictable pattern. From looking at the instantaneous hypervolume, it is noteworthy that *b-CDMORL-d-w* loses ground on cycles 2 and 3 for concept 2, then enters a good pattern only to forget it again.

Experiment 18 (Figure 36) illustrates a somewhat consistent performance improvement in *b-CDMORL-d* and *b-CDMORL-d-w*, while *b-CDMORL* seems to get less certain in the fourth concept, especially in the third cycle (steps 27500-30000).

Overall, the batch-based algorithms appear more capable of learning, but they often do so from a disadvantaged start. The overall accuracy is typically worse than that of the non-batch agents. The reason for this is unclear. It could be that the drift detection is too stable.

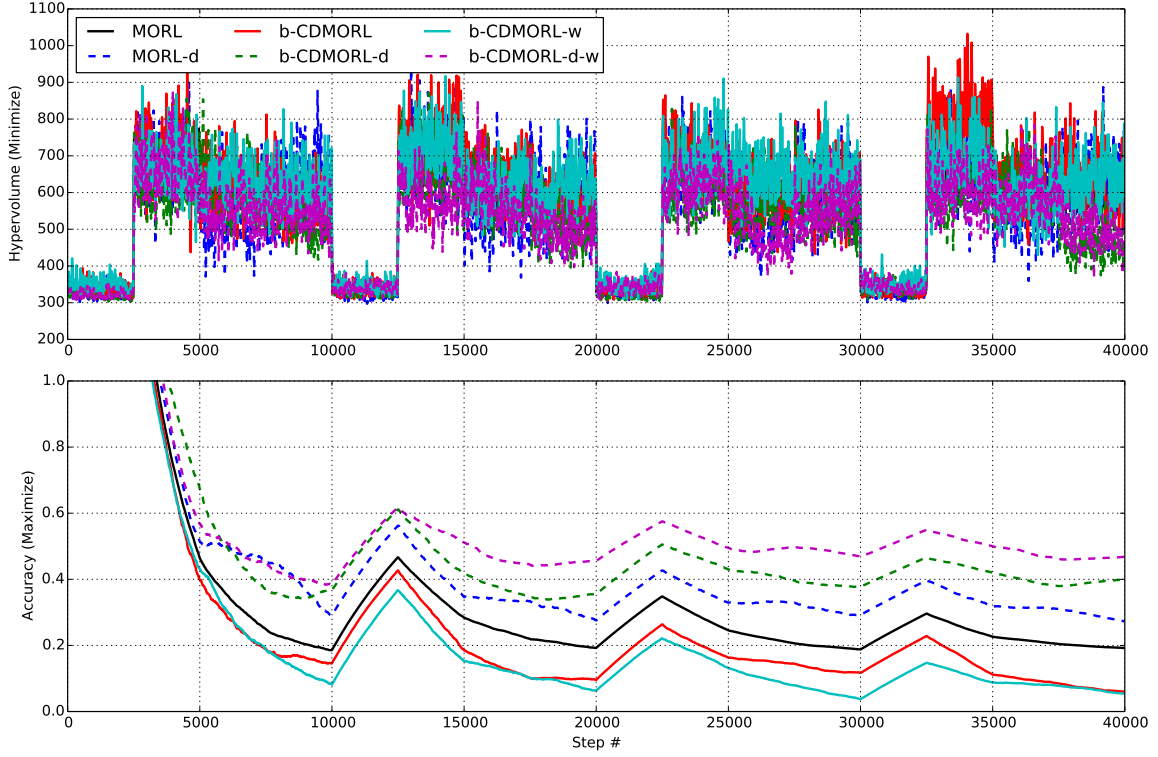


Figure 34: Hypervolume and accuracy for resource gathering experiment 16, comparison among Batch agents and reference agents.

Interestingly, experiment 13 has higher spikes and higher minimum values than the non-batch algorithms. Experiments 15 and 17 also had higher spikes than in the non-batch algorithms. No algorithm in the batch-based set outperformed *MORL-d* across the board. Indeed, *MORL-d* and *MORL* were the worst two algorithms in four of the six experiments for the non-batch algorithms, but this was not the case for the batch algorithms. This is a surprise that possibly merits further parameter tuning in the future.

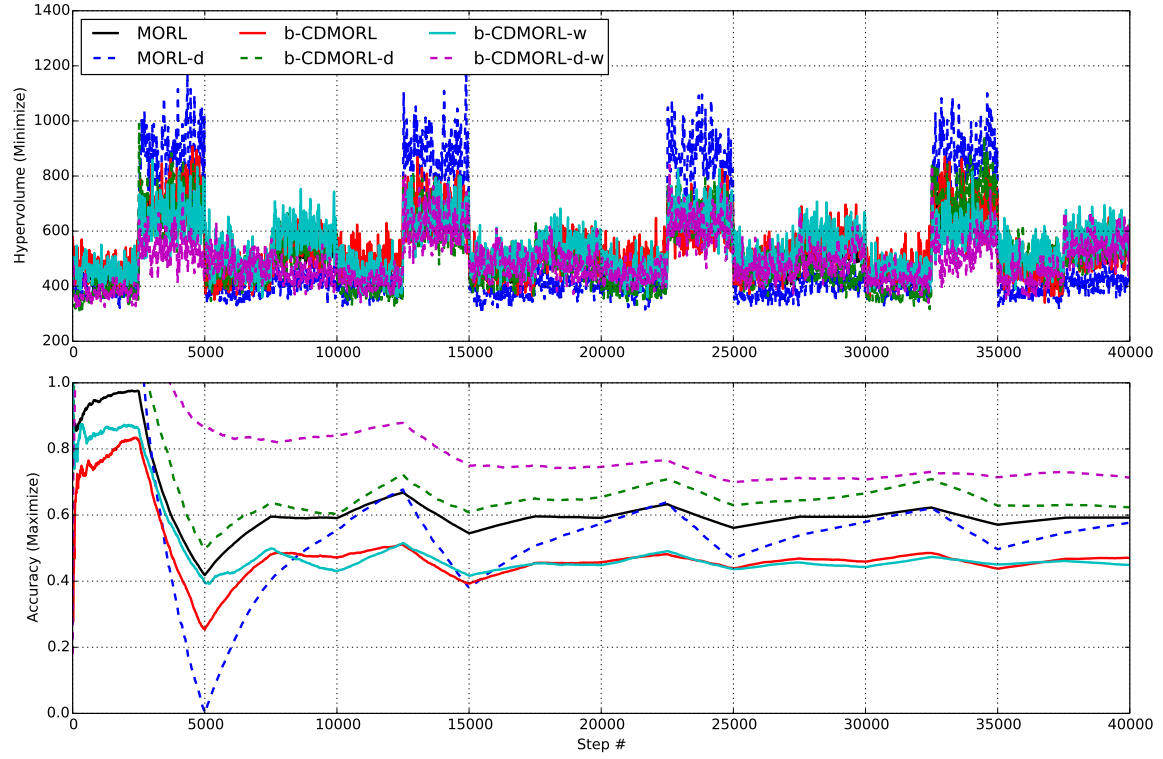


Figure 35: Hypervolume and accuracy for resource gathering experiment 17, comparison among Batch agents and reference agents.

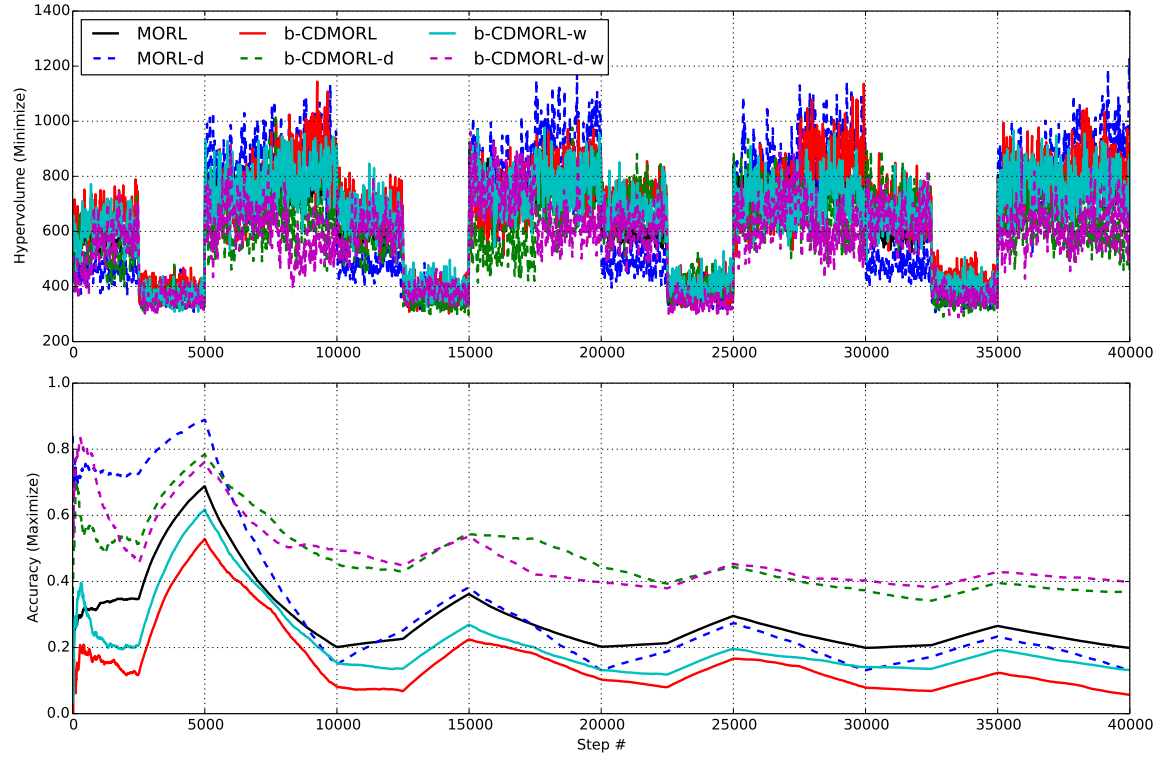


Figure 36: Hypervolume and accuracy for resource gathering experiment 18, comparison among Batch agents and reference agents.

5.3.3 *Accuracy Test: Single Objective Agents.* This section examines the algorithms that are converted to use a single objective. The results are shown in Figures 37-42. Except for the 13th experiment, across the board, the algorithms detecting drift but not using batches performed the best. The single objective reference *SORL* typically performed the worst.

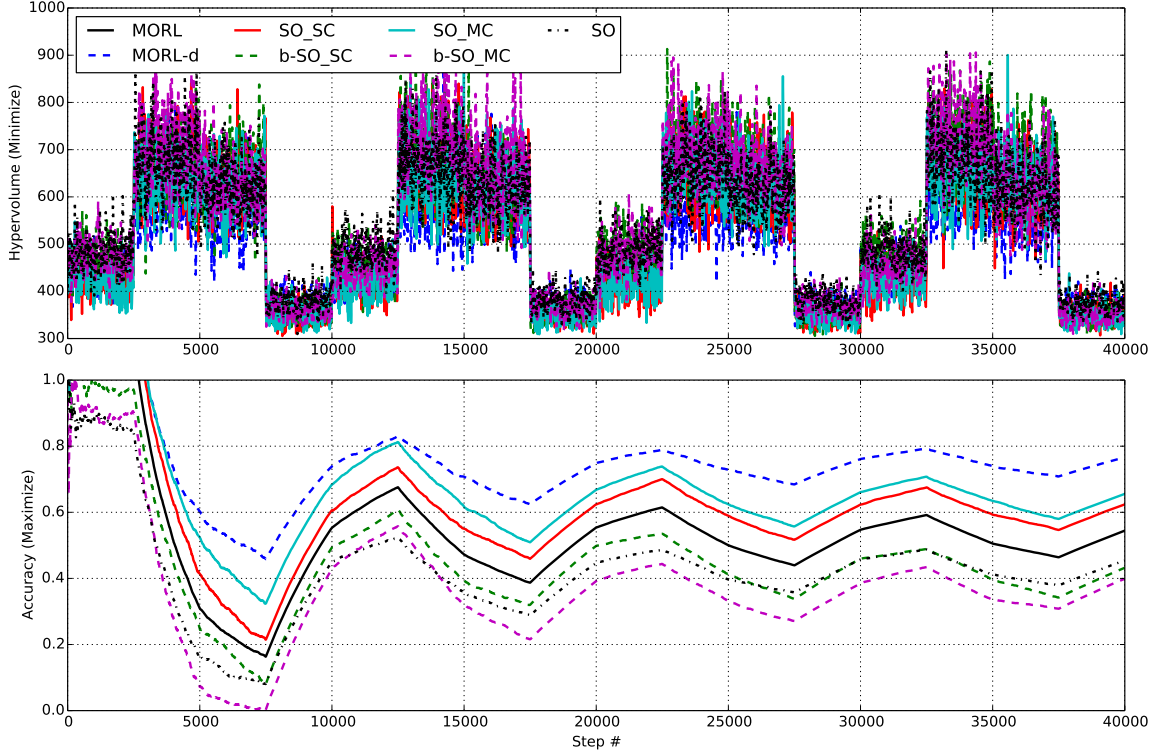


Figure 37: Hypervolume and accuracy for resource gathering experiment 13, comparison among single objective and reference agents.

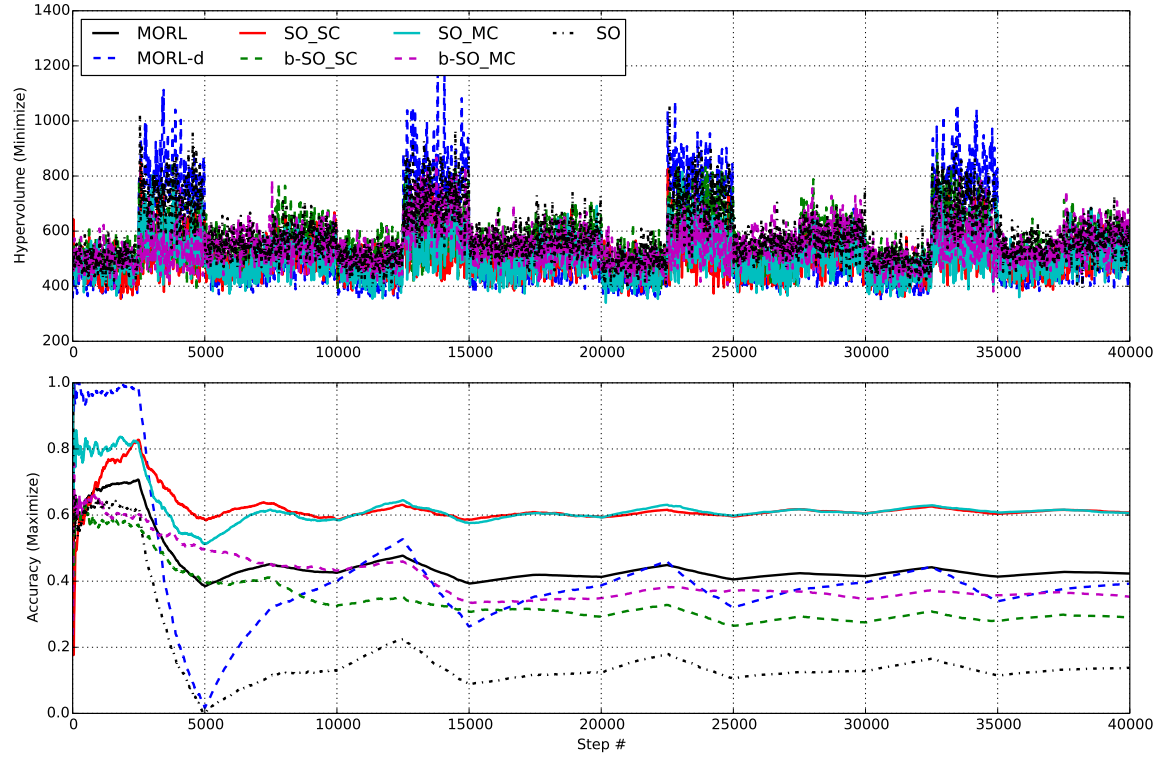


Figure 38: Hypervolume and accuracy for resource gathering experiment 14, comparison among single objective and reference agents.

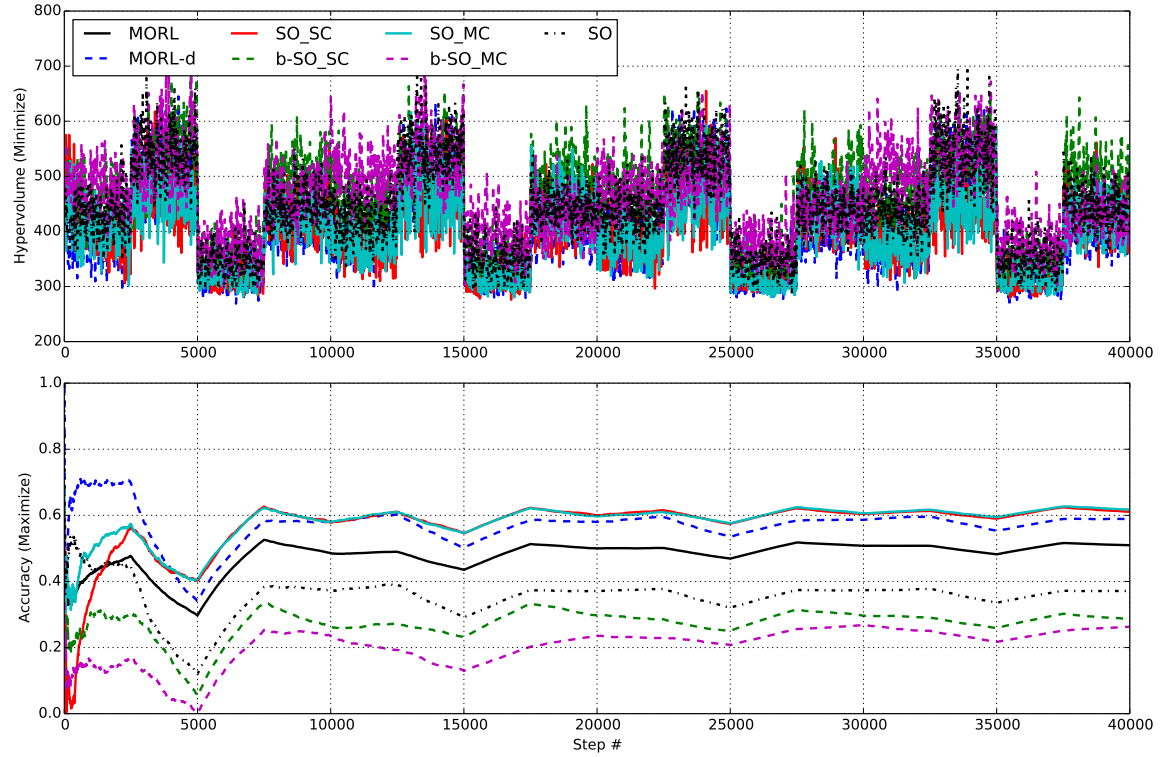


Figure 39: Hypervolume and accuracy for resource gathering experiment 15, comparison among single objective and reference agents.

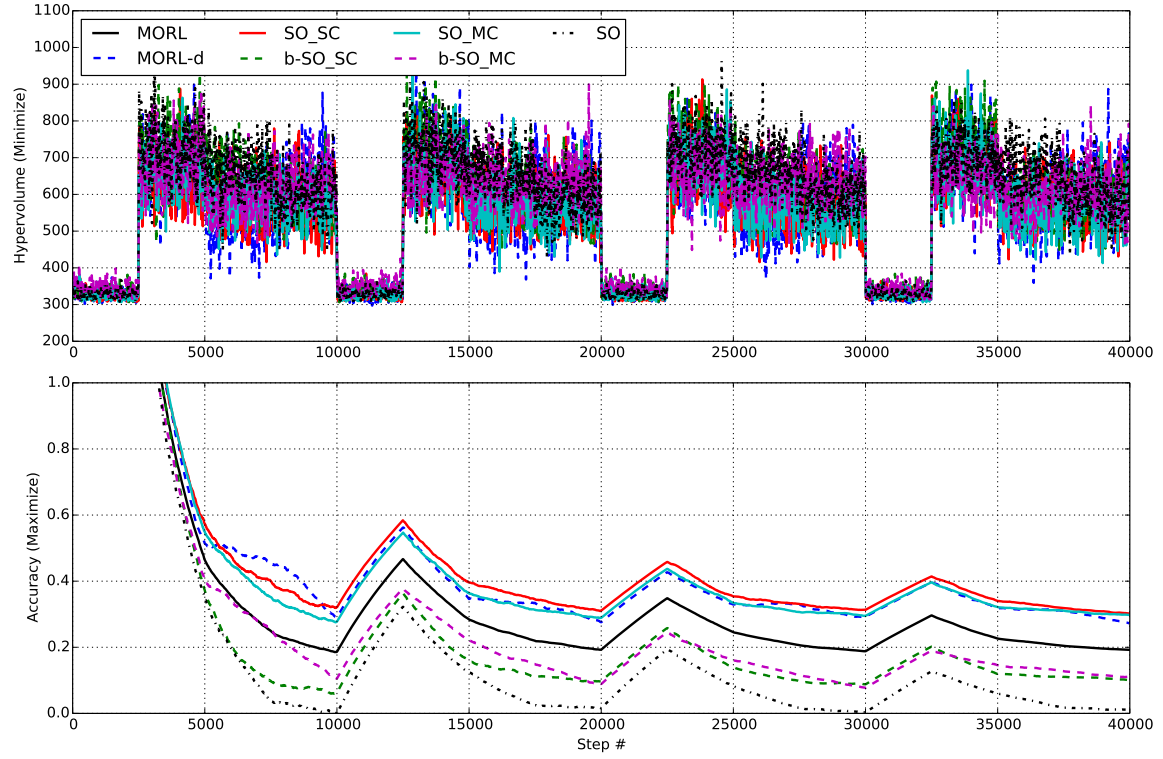


Figure 40: Hypervolume and accuracy for resource gathering experiment 16, comparison among single objective and reference agents.

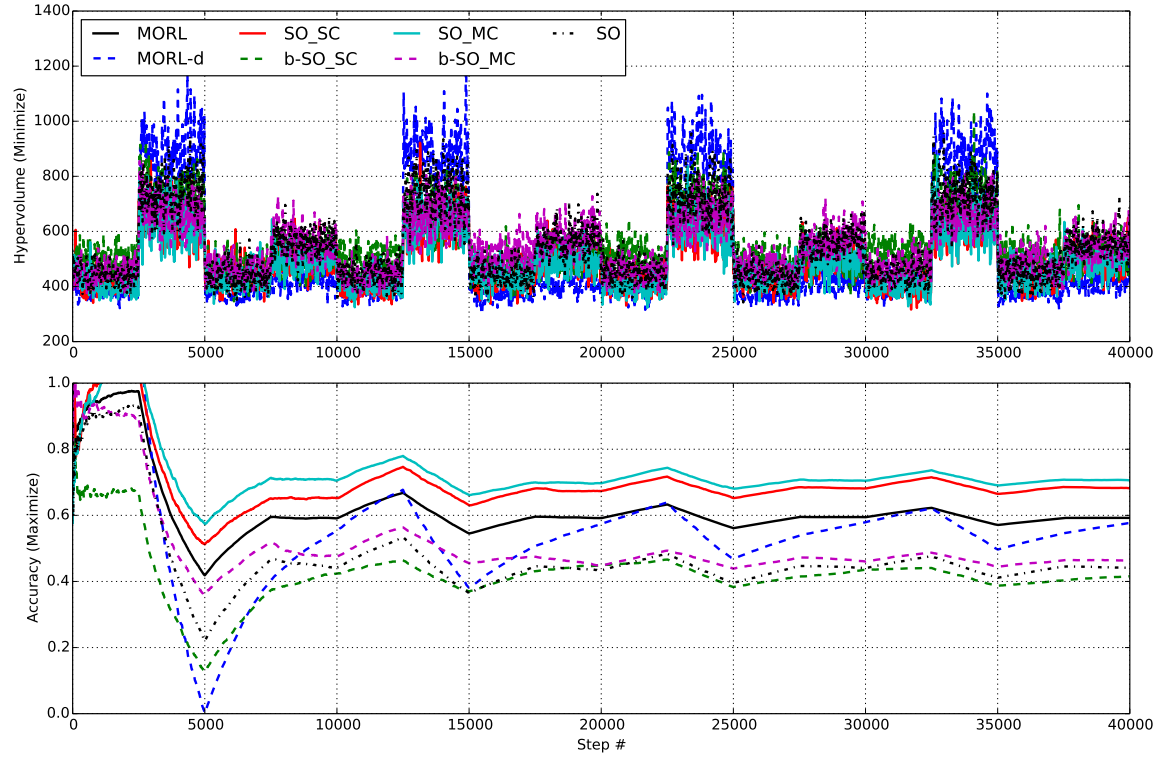


Figure 41: Hypervolume and accuracy for resource gathering experiment 17, comparison among single objective and reference agents.

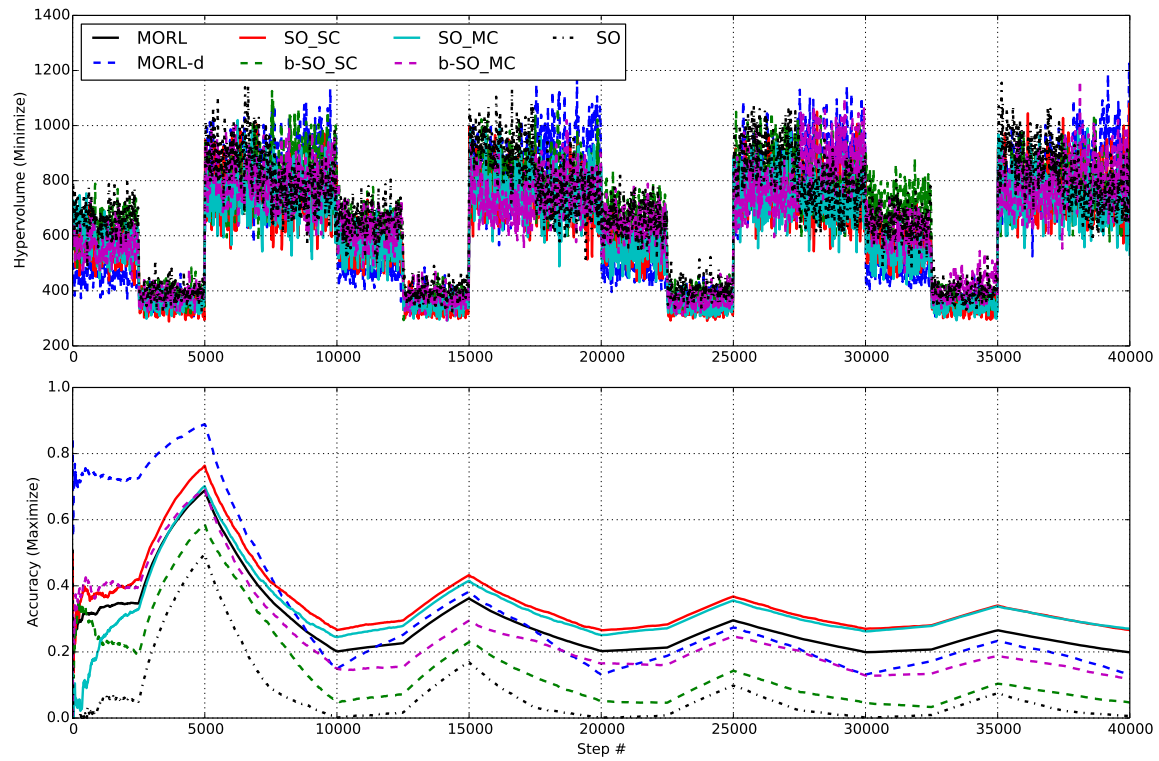


Figure 42: Hypervolume and accuracy for resource gathering experiment 18, comparison among single objective and reference agents.

5.3.4 Summary of Accuracy Tests in Resource Gathering.

This section examines the final performance of the algorithms on the resource gathering domain.

Table 16 shows the final accuracy for the six resource gathering experiments. Every single experiment, *CDMORL-d* and *CDMORL-d-w* outperformed all reference algorithms. Six other algorithms outperformed the reference algorithms in five of six tests: *CDMORL*, *CDMORL-w*, *b-CDMORL-d*, *b-CDMORL-d-w*, *SO_SC*, and *b-SO_MC*. The algorithms optimizing on multiple objectives typically performed substantially better than the single objective algorithms. Of the single objective algorithms, using multiple objectives to detect drift performed better than the algorithms using only a scalarized value to detect drift in eight of twelve cases, and was within noise (0.001) one other time.

Table 16: Summary of ending accuracy for Resource Gathering. For each experiment, the best algorithm in each category and any others within 0.001 are **bold**. The overall best is also *italicized*.

	Experiment Number					
	13	14	15	16	17	18
CDMORL	0.646	0.599	0.603	0.280	0.696	0.273
CDMORL-d	0.802	0.776	0.928	0.577	0.938	0.608
CDMORL-w	0.633	0.608	0.645	0.245	0.686	0.262
CDMORL-d-w	0.798	0.720	0.991	0.518	0.941	0.609
Batch-based CDMORL						
b-CDMORL	0.540	0.379	0.374	0.060	0.470	0.057
b-CDMORL-d	0.937	0.669	0.537	0.400	0.623	0.369
b-CDMORL-w	0.523	0.355	0.232	0.053	0.449	0.132
b-CDMORL-d-w	0.716	0.671	0.605	0.468	0.713	0.399
Single Objective Drift-Aware RL						
SO_SC	0.624	0.607	0.611	0.302	0.682	0.266
b-SO_SC	0.432	0.291	0.287	0.100	0.415	0.047
SO_MC	0.656	0.606	0.618	0.298	0.706	0.271
b-SO_MC	0.399	0.353	0.263	0.109	0.463	0.118
Reference						
MORL	0.545	0.423	0.510	0.192	0.592	0.199
MORL-d	0.766	0.393	0.589	0.272	0.577	0.131
SO	0.454	0.138	0.371	0.010	0.441	0.006

5.3.5 Test for Concept Change Count in Resource Gathering. Table 17

shows the total number of concept changes each algorithm detected. Change detections in this table can result from detecting change or from an initializing algorithm reverting to a recognized prior concept. Overall, the batch-based algorithms were in the right order of magnitude, typically detecting 2-5 times as many changes as there actually were. Non-batch algorithms were two orders of magnitude off, detecting thousands of changes over the forty thousand step experiments - or, one change every 30-40 steps on average. These were mostly due to thrashing of creating concepts, and then getting to a new state and thus initializing, and recognizing a previous concept. It is on purpose that this is possible, to prevent a new state-action pair from being considered concept drift, but perhaps in future work this can be reduced. Though, it is minimized in the batch algorithms.

Table 17: Total number of concept changes that were detected. Average for 30 runs. The correct number is 15. Anything more is a false positive. The answers within 10% of the correct answer are **bold**.

Algorithm	Experiment Number					
	13	14	15	16	17	18
CDMORL						
CDMORL	1289.9	1394.3	1175.9	961.2	1618.1	1551.1
CDMORL-d	1408.8	1557.6	1348.6	1413.6	1654.6	1647.8
CDMORL-w	1236.7	1387.7	1074.5	905.7	1256.8	1595.3
CDMORL-d-w	1419.7	1391.1	1260.9	1369.9	1412.0	1589.0
Batch-based CDMORL						
b-CDMORL	16.4	16.7	33.2	23.7	29.9	31.8
b-CDMORL-d	25.0	40.9	52.1	56.0	57.4	107.3
b-CDMORL-w	15.4	18.6	28.0	20.2	25.1	31.8
b-CDMORL-d-w	33.5	46.8	53.3	52.8	69.1	92.9
Single Objective Drift-Aware RL						
SO_SC	1175.3	1414.8	1086.9	905.3	1291.8	1627.2
b-SO_SC	18.2	20.5	34.7	22.2	25.0	24.1
SO_MC	1184.6	1391.5	1011.7	914.7	1334.1	1663.1
b-SO_MC	13.9	19.9	29.3	21.4	24.3	31.7

Table 18 reports the significance of the false positives. Unlike in the MERLIN experiments, only two experiments showed a result with statistical significance when comparing multi-objective algorithms with single objective algorithms. Also, in contrast with the MERLIN experiments, the significance of decay is not as great. Decay

actually degraded performance in false positives. However, the use of batches made a very significant difference, presumably by mitigating noise.

Table 18: Statistical significance of concept change count by agent category.

	Experiment Number					
	13	14	15	16	17	18
MO Batches	1.424e-39	1.428e-39	1.434e-39	1.431e-39	1.432e-39	1.437e-39
MO Decay	5.515e-03	4.727e-02	4.003e-03	4.420e-08	2.831e-03	3.739e-03
MO WoLF	8.695e-01	7.262e-01	5.765e-01	4.843e-01	3.113e-01	9.042e-01
Drift: M or S	9.442e-01	5.650e-01	6.632e-01	8.095e-01	5.037e-01	6.297e-01
Drift-b: M or S	3.662e-02	5.489e-01	2.047e-01	6.801e-01	7.853e-01	8.555e-02
MO vs SO	6.748e-02	4.391e-01	8.054e-02	4.871e-04	9.959e-03	2.127e-01

5.3.6 Test for Final Concept Count in Resource Gathering. The final number of unique persistent concepts discovered by the classifier are given in Table 19. Overall, most algorithms using either batches and/or decay were within 50% of the correct answer. Algorithms without batches and without decay tended to be off by up to an order of magnitude.

Table 19: Final concept count. Average for 30 runs. The correct answer is 4. Anything within 10% is **bold**.

Algorithm	Experiment Number					
	13	14	15	16	17	18
CDMORL						
CDMORL	12.69	10.62	15.69	11.83	9.83	9.21
CDMORL-d	3.97	4.38	5.48	4.83	4.69	5.45
CDMORL-w	14.00	10.90	14.55	11.14	10.45	8.76
CDMORL-d-w	3.79	4.17	4.90	4.21	4.31	5.41
Batch-based CDMORL						
b-CDMORL	3.28	3.79	4.59	3.90	4.48	4.10
b-CDMORL-d	4.45	5.83	6.34	6.45	6.90	10.55
b-CDMORL-w	3.24	3.86	3.86	3.66	4.07	4.21
b-CDMORL-d-w	4.69	5.76	6.76	5.93	7.24	8.72
Single-Objective Drift-Aware RL						
SO_SC	12.90	10.34	15.14	11.66	10.83	9.59
b-SO_SC	3.52	3.93	4.31	3.79	3.90	3.93
SO_MC	12.93	11.45	15.76	11.17	9.52	9.28
b-SO_MC	3.10	3.55	3.66	3.66	3.93	4.38

Table 20 shows the significance of the final concept count results. Decay had a significant result, which is interesting because it improved CDMORL without batches but degraded CDMORL with batches.

Table 20: Statistical significance of final concept counts by agent category.

Algorithm	Experiment Number					
	13	14	15	16	17	18
MO Batches	4.674e-12	3.965e-08	2.692e-11	4.570e-09	8.796e-05	3.289e-03
MO Decay	3.385e-05	3.016e-04	6.207e-04	8.592e-03	3.896e-03	8.431e-02
MO WoLF	7.072e-01	8.236e-01	3.684e-01	1.502e-01	7.990e-01	4.222e-01
Drift: M or S	9.562e-01	5.782e-02	6.732e-01	4.274e-01	3.442e-02	5.231e-01
Drift-b: M or S	6.893e-02	1.897e-01	2.543e-02	7.102e-01	9.479e-01	2.979e-01
MO vs SO	1.631e-02	4.293e-02	1.053e-01	1.459e-01	4.747e-01	7.728e-01

5.4 Overall Analysis of CDMORL

This section reviews the measures taken of the agents over the experiments to answer the four research questions.

Does CDMORL have greater accuracy than MORL, with respect to hypervolume?

According to the *accuracy* measure, the four CDMORL algorithms with decay outperformed the baseline MORL and MORL-d algorithm 66 out of 72 times. For the two CDMORL-d algorithms not using batches, CDMORL-d-w was outperformed just once on a stationary MERLIN test, and CDMORL-d was never outperformed. The algorithms without decay performed poorly, outperforming MORL or MORL-d less than half the time - 28 of 72 trials.

CDMORL with decay consistently has greater accuracy than MORL.

These results could be skewed against batches because the batch size didn't match the concept change size in non-stationary MERLIN and because both MERLIN experiments were deterministic, making 100% accuracy feasible.

Batch-based methods may be ill-suited to resource gathering, because the only indicators of drift are when the enemies change position or the treasures change position. The sparsity of these indicators could mean that only one objective in one sample in a batch contains the information needed to identify drift, and thus it could be easily outvoted. The success of batches is possibly correlated to how close the enemies or treasures were to the spawn location, which could result in multiple indications per batch.

Which concept drift handling techniques are most accurate?

As discussed above, **CDMORL-d** is most accurate, followed by **CDMORL-d-w**, followed by the batch-based version of these two. MORL-d consistently outperformed the CDMORL methods not using decay. This underscores the importance of converging within a concept.

WoLF generally made little to no difference, except for b-CDMORL-d-w, where WoLF made a minor improvement in accuracy. This is possibly due to defending against over-reactions. WoLF made little difference in the non-batch based algorithms.

Single objective methods consistently underperformed CDMORL methods using decay in accuracy. The best single objective methods were on par with the worst CDMORL methods and better than the worst batch-based CDMORL methods.

Which multi-objective concept drift strategy most accurately identifies the correct number of concept changes?

With high statistical significance, algorithms using **batches** most accurately identified the correct number of concept changes, as did algorithms with **decay**. The **combination of batches and decay** is synergistic.

Which multi-objective concept drift strategy most accurately identifies the correct number of concepts?

With high statistical significance, algorithms using **batches** most accurately identified the correct number of concepts, as did algorithms with **decay**. Like with detecting concept changes, the **combination of batches and decay** is synergistic.

Due to the nature of batches, they had $\frac{1}{N_B}$ of the opportunities to detect drift as non-batch methods, which may indicate that noise rejection of even one step is valuable.

WoLF made no difference in false positive count or in concept count.

5.5 Discussion

This section discusses the overall findings of the experiments followed by observations made in the course of research and experimentation, but are not main contributions of this dissertation or are not substantiated through data. It discusses two convergence matters, future emphasis (γ), two matters in concept classification, batches, another metric for future consideration, and other formulations for MDPs.

The best algorithm overall is CDMORL-d. This is because it had a competitive accuracy value, typically in the top two performers. When it was outperformed, it was always by one or two CDMORL agents with decay. It was the highest performer 7 out of 18 experiments (one of which was a virtual tie in MERLIN non-stationary #11). This best-overall also considers that it never had a truly bad accuracy (unlike b-CDMORL-d-w). It was in the second tier of performers for false positive detection and concept count. Relative to the non-decay counterparts, the other three CDMORL algorithms with decay are also strong performers in accuracy, false positive detection, and concept count.

CDMORL-d and CDMORL-d-w consistently had the best accuracy, sometimes beaten by one of the other decay algorithms. In fact, there was never a time in 18 experiments that a non-decay algorithm achieved the best accuracy (5 had decay, 10 did not).

The best false positive counts and concept counts were by batch-based algorithms and decayed algorithms, making b-CDMORL-d and b-CDMORL-d-w the best overall, followed by algorithms with batches or decay but not both.

Beyond the results identified from analyzing the data presented in this dissertation, a number of other observations and inferences were identified in the course of this research. First, the utility of using a *CDMORL* algorithm depends on the expected useful life of a reference algorithm with respect to using drift detection. When all possible concepts are experienced before the learning and exploration rate have completely decayed, *MORL-d* is quite competitive. However, when that assumption is violated, *MORL-d* achieves poor accuracy due to concept drift.

Next, developing a robust characterization requires the agent to experience a sufficient number of steps. Characterization convergence is necessary for robust concept detection, which is in turn needed for concept drift characterization and thus drift detection and concept recognition. The step and state-action pair based thresholds were designed to reduce this problem. The thresholds were set at approximately

$\frac{|S||A|}{10}$ state-action pairs and $|S||A|$ steps as an alternate measure. Another essential part of convergence is handling each state-action pair separately for each concept. Earlier attempts tracked either visits to a state (even though rewards are a function of action as well) or just total steps taken. Earlier attempts also failed to recognize that unvisited or unconverged information is unreliable for concept characterization, leading to false positives and missed hits. Accordingly, many parts of the algorithm rely on detail at the state-action pair level.

Next, the future discount parameter γ adjusts how much the future reward is emphasized. This is problematic for convergence-based concept drift detection if a given state-action pair has reached convergence, but an un-converged state-action pair becomes the new optimum. For example, the utility of being killed by an enemy or discovering a treasure is a very large change with respect to the surrounding reward landscape. This frequently creates a large change to $\vec{Q}(s', a')$ that exceeds the drift threshold $\theta_2 = 3\alpha(s, a)$ for the current $\vec{Q}(s, a)$. A high γ makes convergence difficult and a low γ makes it harder to develop an optimal behavior. Perhaps in future work the $\gamma\vec{Q}(s', a')$ term can be set to have a maximum change impact, or perhaps γ can be a function of how stable the concept is believed to be or of convergence for all involved state-action pairs.

The concept drift voting mechanism is helpful by isolating the impact of a single objective with respect to others. A downside is that there are vote threshold parameters to set and, further, if only a single objective is impacted, such as damage from an enemy or treasure, it still gets only a singular vote even if the concept change is otherwise decisive. Three other concept drift methods were investigated in this research. Two are discussed briefly in the appendix. The third was to sum, for all objectives and in a batch for all samples, the absolute value of all best ratios used in concept drift detection (see Alg 8 line 10) and divide by $\alpha(s, a)$, and find the average, then detect drift using a drift threshold of more than 2 and a warning threshold of more than 1. All three other methods were less reliable than the method presented in Chapter III. It would be preferable to find a method that uses fewer parameters

that could form a theoretical or mathematical basis for drift detection. However, this work is a first step in the direction of concept drift-aware MORL and so this is left for future work.

To reduce the likelihood of having multiple concepts claiming to be stable at the same time during a recognition attempt, new concepts are created with a clean, randomized Q and π . This worked well in these domains, where most rewards were close to 0 and many actions were approximately as good in others. But in more challenging domains, cloning tables and perhaps adding noise (perhaps, 2α to Q table and $\delta_{winning}$ to π) would facilitate making the agent comparably competitive with traditional methods, if not better, as the agent starts up and begins converging concept characterizations.

The batch-based algorithms were best overall at handling new environments and new drift detection mechanisms. Using the mechanisms in Appendices A and B, b-CDMORL-d was always the best performer. Once the final drift characterization method was designed and once the environments were tuned, and given the types of environments, it became less reliable at accuracy than the non-batch methods. The batch-based algorithms with decay reported the fewest false positives and the most precise concept count (as mentioned before, reducing γ to or near to 0 causes all algorithms to achieve near perfect results on both metrics).

Future work should consider measures of the lifespan of concepts and the types of concept change detected. From the results in resource gathering, it appears that having more false positives was not necessarily negative. Further, recognitions are counted the same as new initializations. An agent with numerous false positives with many more initializations than recognitions would have less time learning than an agent that had more recognitions. Even with thrashing between similar concepts, recognitions are for permanent concepts, and so the information learned is retained and thus not catastrophically forgotten.

One formulation investigated for resource gathering was to make enemy attacks, enemy misses, treasure grabs, and treasure shortfalls the only rewards, thereby giving sparse and delayed rewards. However, how this would interact with the batch classifiers was uncertain, and so this approach was ultimately abandoned for the version presented earlier, which added small punishments to each state.

5.6 Summary

Experimental results demonstrated that an agent aware of concept drift in drifting environments consistently outperforms an agent unaware of concept drift. In environments without concept drift, *CDMORL* agents performed similarly to *MORL* agents, which means that *MORL* experiences performance degradation under concept drift.

The *CDMORL-d* and *CDMORL-d-w* agents consistently outperformed other all algorithms. The combination of *CDMORL* and decay enabled an autonomous agent to characterize, detect, and recognize change while using an increasingly converged concept definition with an increasingly precise and behavior definition. Eight of the twelve drift detection algorithms outperformed all reference algorithms consistently. The batch algorithms always had the best classifier performance, and why they did not end up performing the best is unclear. Algorithms with decay tended to perform the best overall and algorithms with WoLF suffered the least in complex concept changes.

Overall, this work successfully contributed a successful set of solutions to non-stationary multi-objective problems.

VI. Conclusion

Reinforcement learning agents generally leverage the Markov assumption that the current state is all an agent needs to know to achieve optimal behavior. However, in more real-world environments, this is not the case, as partially observable changes in the form of concept drift cause the meaning of agent observations to change. To enable an autonomous agent to handle this kind of environment, it must have a way to characterize an environment’s concept, detect concept drift, and recognize previous concepts. Because more practical agents handle multiple tasks simultaneously, this integration was achieved with multi-objective reinforcement learning, which has the added benefit of providing additional information the agent can use to characterize the concept.

This dissertation presented a novel algorithm integrating concept drift learning into multi-objective reinforcement learning along with three different techniques. The developed agent characterizes concepts by scoring the ratio between the relative change in q and the value of α using the value from each of its objectives. With this characterization, the agent defines concepts, detects drift, and recognizes prior concepts. The agent uses an initialization characterization that reduces the number of concepts based on noise and makes it easier to recover from reacting to such noise. The agent and its augmentations were tested on theoretical domain MERLIN as well as the Resource Gathering domain, which is representative of strategy games. This chapter first presents the conclusions regarding this research. This is followed by future research areas.

6.1 Conclusions

The presented CDMORL, a concept drift-aware multi-objective reinforcement learning agent, mitigates catastrophic forgetting and the plasticity-stability dilemma in reinforcement learning (RL). This algorithm is a first-of-its-kind algorithm, being truly multi-objective (learning multiple q values for each state-action pair) and integrating concept drift learning. Results demonstrate that agents using this algorithm

consistently outperform agents that assume a stationary environment, both in a deterministic MOMDP environment such as MERLIN as well as the stochastic resource gathering environment, which also has a few extreme feedback signals characteristic of other RL problems. This section discusses the contributions of each algorithm variant: decaying rates, variable learn rates (WoLF), and batches.

6.1.1 Decay. The notion of a decaying learning rate was originally investigated as a form of a straw man, except it is popular in literature as a remedy to the plasticity-stability dilemma (generally in environments that are stationary but stochastic). The *MORL-d* algorithm performs well in terms of accuracy in the stationary MERLIN tests (Table 6). The CDMORL algorithms using decay were more accurate in terms of hypervolume. They also had among the best final concept counts.

6.1.2 Dynamic Learning Rates with WoLF. The Win or Learn Fast (WoLF) agents appear to blunt the negative impact of complex concept changes. Overall, the use of WoLF does not appear to have had an impact in accuracy in hypervolume. It also did not have a significant result in concept change count or final concept count.

6.1.3 Batches. The use of batches greatly improved the ability of the classifier to get close to detecting the correct number of concept changes and to identify the correct number of persistent, unique concepts. During experimentation with other concept classification methods and before convergence parameters had been tuned, batches were without exception the most reliable way of compensating for sub-optimal conditions. In those trials, batch-based agents consistently outperformed all others in all measures. However, in the tuned environments, they are sub par in terms of accuracy with respect to hypervolume. However, batch-based methods had the most accurate concept change counts and final concept counts.

6.1.4 Combinations. The overall best algorithm was *CDMORL-d* because it reliably achieved the best or second best accuracy and had reasonable false positive

and concept count performance. The combination of decay with batches gave the best results in terms of concept change count and final concept count. Algorithms with WoLF performed comparably well as those that did not have it. The WoLF algorithms typically had neither the exceptionally poor nor the exceptionally strong instantaneous hypervolume as other algorithms, and so WoLF could typically be viewed as safer.

6.2 Future Work

Two future work ideas were explicitly suggested earlier in Section 5.5. Those ideas include:

- Examine methods of optimally engaging the future discount parameter γ without creating false concept drift reports while converging to an optimal utility function representation.
- Reduce the number of parameters used in concept characterization, possibly while simplifying the overall process to make theoretical analysis simpler.

Formal analysis of parameterizations: As discussed in Section 5.5, *CD-MORL* requires a certain amount of initialization to be competitive, and *MORL-d* is able to make adjustments to be competitive within a certain span. It may be beneficial to explore, based on the expected lifespan of the agent and the expected characteristics of the environment when one method is a better choice than the other.

Improved concept classification metrics: The concept classification metrics used in this dissertation are less informative than other metrics could be. For example, a confusion matrix could be adapted to a temporal domain to illustrate which agent-generated concepts mapped to which environment truth concepts at a given time.

Non-stationary MORL testbed: The combination of the contributed CD-MORL algorithm and the test environment provides a good platform to research non-stationarity. Source code from this research could form the basis of a new standardized test bed.

6.3 *Impact*

CDMORL is a first-of-its-kind enabler for autonomy. It has application to personalized learning, autonomous research systems used in materials and manufacturing research, as well as potential application in cognitive modeling.

Personalized learning in the form of an intelligent tutoring system is able to monitor student performance. In lieu of other vigilance or fatigue monitoring that may not be practical in a given situation (i.e. eye tracking, heart rate monitors, etc), the agent can theoretically identify the reduction in performance and seek to improve student performance, albeit at a degraded level, and then revert to faster paced techniques when the student becomes more engaged.

Materials and manufacturing research, such as work into carbon nanotubes, is able to optimize agent predictions of characteristics of carbon nanotube growth. However, when predictions begin to be inconsistent, this is an indication that something may have happened: a very minor earth quake, a gas or water supply running dry, a change in laser focus, and so on. While waiting for human intervention, the agent could continue to learn relative patterns, or otherwise stop until changes have been made.

Cognitive modeling studies into non-stationary environments investigate varying levels of cognition, including fine motor skills such as sensor operation of a remotely piloted asset.

In each of these fields, this research provides new opportunity to push the state of the art in that application.

Bibliography

1. Adams, Ryan Prescott and David JC MacKay. “Bayesian online changepoint detection”. *arXiv preprint arXiv:0710.3742*, 2007. URL <https://arxiv.org/pdf/0710.3742.pdf>.
2. Baena-Garcia, Manuel, José del Campo-Ávila, Raúl Fidalgo, Albert Bifet, Ricard Gavalda, and Rafael Morales-Bueno. “Early drift detection method”. *Fourth International Workshop on Knowledge Discovery from Data Streams*, volume 6, 77–86. 2006.
3. Barrett, Leon and Srinu Narayanan. “Learning all optimal policies with multiple criteria”. *Proceedings of the 25th International Conference on Machine Learning*, 41–47. ACM, 2008.
4. Berro, Alain and Yves Duthen. “Search for Optimum in Dynamic Environment: a Efficient Agent-Based Method”. *Genetic and Evolutionary Computation Conference. Workshop Program*, 51–54. 2001.
5. Bifet, Albert and Ricard Gavalda. “Learning from time-changing data with adaptive windowing”. *Proceedings of the 2007 SIAM International Conference on Data Mining*, 443–448. 2007.
6. Bowling, Michael and Manuela Veloso. “Multiagent Learning using a Variable Learning Rate”. *Artificial Intelligence*, 136(2):215–250, 2002.
7. Brockhoff, Dima, Tobias Wagner, and Heike Trautmann. “On the Properties of the R2 Indicator”. *Proceedings of the Fourteenth International Conference on Genetic and Evolutionary Computation Conference*, 465–472. 2012.
8. Calandra, Roberto, Tapani Raiko, Marc Peter Deisenroth, and Federico Montesino Pouzols. “Learning deep belief networks from non-stationary streams”. *International Conference on Artificial Neural Networks*, 379–386. 2012.
9. Cassandra, Anthony R, Leslie Pack Kaelbling, and Michael L Littman. “Acting optimally in partially observable stochastic domains”. *AAAI-94 Proceedings*, 1023–1028. 1994.
10. Chatterjee, Krishnendu, Rupak Majumdar, and Thomas A Henzinger. “Markov decision processes with multiple objectives”. *Proceedings of the Symposium on Theoretical Aspects of Computer Science*, 325–336. Springer, 2006.
11. Choi, Samuel PM, Dit-Yan Yeung, and Nevin L Zhang. “Hidden-Mode Markov Decision Processes for Nonstationary Sequential Decision Making”. *Sequence Learning*, 264–287. Springer, 2001.
12. Coello, Carlos Coello, Gary B Lamont, and David A Van Veldhuizen. *Evolutionary Algorithms for Solving Multi-Objective Problems*. Springer, 2007.

13. Cruz, Carlos, Juan R González, and David A Pelta. “Optimization in Dynamic Environments: a Survey on Problems, Methods and Measures”. *Soft Computing*, 15(7):1427–1448, 2011.
14. Deb, Kalyanmoy and Himanshu Jain. “An evolutionary many-objective optimization algorithm using reference-point based non-dominated sorting approach, part I: Solving problems with box constraints”. *IEEE Transactions on Evolutionary Computation*, 2013.
15. Deon, Garrett. “Random Problem Generator for Multi-Task Reinforcement Learning Problems”, October 2015. URL <https://github.com/deong/merlin>.
16. Ditzler, Gregory, Manuel Roveri, Cesare Alippi, and Robi Polikar. “Learning in nonstationary environments: A survey”. *IEEE Computational Intelligence Magazine*, 10(4):12–25, 2015.
17. Doya, Kenji, Kazuyuki Samejima, Ken-ichi Katagiri, and Mitsuo Kawato. “Multiple Model-Based Reinforcement Learning”. *Neural Computation*, 14(6):1347–1369, 2002.
18. Er, M.J. and C. Deng. “Online tuning of fuzzy inference systems using dynamic fuzzy Q-learning”. *IEEE Transactions on Systems, Man, and Cybernetics*, 34(3):1478–1489, 2004.
19. Er, M.J. and L. San. “Automatic generation of Fuzzy Inference Systems using Incremental-Topological-Preserving-Map-based Fuzzy Q-Learning”. *Fuzzy Systems. (IEEE World Congress on Computational Intelligence).*, 467–474. 2008.
20. Farina, Marco, Kalyanmoy Deb, and Paolo Amato. “Dynamic multiobjective optimization problems: test cases, approximations, and applications”. *IEEE Transactions on Evolutionary Computation*, 8(5):425–442, 2004.
21. Fearnhead, Paul and Zhen Liu. “On-line inference for multiple changepoint problems”. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 69(4):589–605, 2007.
22. Gama, João, Pedro Medas, Gladys Castillo, and Pedro Rodrigues. “Learning with drift detection”. *Brazilian Symposium on Artificial Intelligence*, 286–295. Springer, 2004.
23. Gama, João, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. “A Survey on Concept Drift Adaptation”. *ACM Computing Surveys (CSUR)*, 46(4):44, 2014.
24. Garnett, Roman, Michael A Osborne, and Stephen J Roberts. “Sequential Bayesian prediction in the presence of changepoints”. *Proceedings of the 26th Annual International Conference on Machine Learning*, 345–352. ACM, 2009.
25. Garrett, Deon, Jordi Bieger, and Kristinn R Thórisson. “Tunable and generic problem instance generation for multi-objective reinforcement learning”. *IEEE*

Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL), 1–8. 2014.

26. Goldberg, Dani and Maja J Matarić. “Maximizing Reward in a Non-Stationary Mobile Robot Environment”. *Autonomous Agents and Multi-Agent Systems*, 6(3):287–316, 2003.
27. Gonçalves Jr, Paulo Mauricio and Roberto Souto Maior De Barros. “RCD: A recurring concept drift framework”. *Pattern Recognition Letters*, 34(9):1018–1025, 2013.
28. Hansen, Michael Pilegaard and Andrzej Jaskiewicz. *Evaluating the quality of approximations to the non-dominated set*. Technical report, 1998.
29. Harries, Michael Bonnell, Claude Sammut, and Kim Horn. “Extracting Hidden Context”. *Machine Learning*, 32(2):101–126, 1998.
30. Hartert, Laurent and Moamar Sayed-Mouchaweh. *Semisupervised Dynamic Fuzzy K-Nearest Neighbors*, chapter 5, 103–124. Learning in Non-Stationary Environments. Springer, 2012.
31. Hester, Todd and Peter Stone. “TEXPLORE: real-time sample-efficient reinforcement learning for robots”. *Machine Learning*, 90(3):385–429, 2013.
32. Jin, Yaochu and Bernhard Sendhoff. “Constructing dynamic optimization test problems using the multi-objective optimization concept”. *Applications of Evolutionary Computing*, 525–536. Springer, 2004.
33. Kaelbling, Leslie Pack, Michael L Littman, and Anthony R Cassandra. “Planning and Acting in Partially Observable Stochastic Domains”. *Artificial Intelligence*, 101(1):99–134, 1998.
34. Kaelbling, Leslie Pack, Michael L Littman, and Andrew W Moore. “Reinforcement Learning: A Survey”. *Journal of Artificial Intelligence Research*, 1996.
35. Kamaya, Hiroyuki, Haeyeon Lee, and Kenichi Abe. “Switching Q-learning in partially observable Markovian environments”. *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 2, 1062–1067. 2000.
36. Kearns, Michael and Satinder Singh. “Finite-Sample Convergence Rates for Q-Learning and Indirect Algorithms”. *Advances in Neural Information Processing Systems*, 1999.
37. Kolter, J Zico and Marcus A Maloof. “Dynamic weighted majority: An ensemble method for drifting concepts”. *The Journal of Machine Learning Research*, 8:2755–2790, 2007.
38. Kolter, Jeremy Z and Marcus A Maloof. “Dynamic weighted majority: A new ensemble method for tracking concept drift”. *Third IEEE International Conference on Data Mining*, 123–130. 2003.

39. Kruskal, William H and W Allen Wallis. “Use of ranks in one-criterion variance analysis”. *Journal of the American Statistical Association*, 47(260):583–621, 1952.
40. Kuncheva, Ludmila I. “Classifier ensembles for changing environments”. *Multiple Classifier Systems*, 1–15. Springer, 2004.
41. Lazarescu, Mihai. “A Multi-Resolution Learning Approach to Tracking Concept Drift and Recurrent Concepts.” *Pattern Recognition in Information Systems*, 52. 2005.
42. Lazarescu, Mihai M, Svetha Venkatesh, and Hung H Bui. “Using multiple windows to track concept drift”. *Intelligent Data Analysis*, 8(1):29–59, 2004.
43. Leon-Garcia, Alberto. *Probability and Random Processes for Electrical Engineering*. Addison Wesley-Longman, second edition, 1994.
44. Li, Pei-Pei, Xindong Wu, and Xuegang Hu. “Mining Recurring Concept Drifts with Limited Labeled Streaming Data.” *ACML*, 241–252. 2010.
45. Liao, Xuejun and Lawrence Carin. “Migratory logistic regression for learning concept drift between two data sets with application to UXO sensing”. *IEEE Transactions on Geoscience and Remote Sensing*, 47(5):1454–1466, 2009.
46. Littman, Michael L. “A tutorial on partially observable Markov decision processes”. *Journal of Mathematical Psychology*, 53(3):119–125, 2009.
47. Mannor, Shie and Nahum Shimkin. “A geometric approach to multi-criterion reinforcement learning”. *The Journal of Machine Learning Research*, 5:325–360, 2004.
48. Nguyen, Trung Thanh, Shengxiang Yang, and Juergen Branke. “Evolutionary Dynamic Optimization: A Survey of the State of the Art”. *Swarm and Evolutionary Computation*, 6:1–24, 2012.
49. Obayashi, Masanao, Takashi Kuremoto, and Kunikazu Kobayashi. “A self-organized fuzzy-neuro reinforcement learning system for continuous state space for autonomous robots”. *International Conference on Computational Intelligence for Modelling Control & Automation*, 551–556. 2008.
50. Osborne, Michael A, Roman Garnett, and Stephen J Roberts. “Active data selection for sensor networks with faults and changepoints”. *IEEE International Conference on Advanced Information Networking and Applications (AINA)*, 533–540. 2010.
51. Oubbati, Mohamed, Timo Oess, Christian Fischer, and Günther Palm. “Multi-objective reinforcement learning using adaptive dynamic programming and reservoir computing”. *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*. 2012.
52. Reddy, M Janga and D Nagesh Kumar. “Multiobjective differential evolution with application to reservoir system optimization”. *Journal of Computing in Civil Engineering*, 21(2):136–146, 2007.

53. Roijers, Diederik Marijn, Peter Vamplew, Shimon Whiteson, and Richard Dazeley. “A survey of multi-objective sequential decision-making”. *arXiv preprint arXiv:1402.0590*, 2014.
54. Sayed-Mouchaweh, Moamar and Edwin Lughofer. *Learning in Non-Stationary Environments: Methods and Applications*. Springer DE, 2012.
55. Schlimmer, Jeffrey C and Richard H Granger Jr. “Incremental Learning from Noisy Data”. *Machine Learning*, 1(3):317–354, 1986.
56. Schmidhuber, Jürgen. “Reinforcement learning in Markovian and non-Markovian environments”. *Advances in Neural Information Processing Systems*, 500–506. 1991.
57. Schmidhuber, Jürgen. “Deep learning in neural networks: An overview”. *Neural Networks*, 61:85–117, 2015.
58. Shelton, Christian Robert. *Importance sampling for reinforcement learning with multiple objectives*. Ph.D. thesis, 2001.
59. da Silva, ALCB Bruno Castro, Denise de Oliveria, and EW Basso. “Adaptive traffic control with reinforcement learning”. *Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 80–86. 2006.
60. da Silva, Bruno C, Eduardo W Basso, Ana LC Bazzan, and Paulo M Engel. “Dealing with Non-Stationary Environments Using Context Detection”. *Proceedings of the 23rd International Conference on Machine learning*. ACM, 2006.
61. Stuart Russell, Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, second edition, 2002.
62. Sugiyama, Masashi, Matthias Krauledat, and Klaus-Robert Müller. “Covariate shift adaptation by importance weighted cross validation”. *The Journal of Machine Learning Research*, 8:985–1005, 2007.
63. Sugiyama, Masashi, Shinichi Nakajima, Hisashi Kashima, Paul V Buenau, and Motoaki Kawanabe. “Direct importance estimation with model selection and its application to covariate shift adaptation”. *Advances in Neural Information Processing Systems*, 1433–1440. 2008.
64. Sun, Yange, Zhihai Wang, Haiyang Liu, Chao Du, and Jidong Yuan. “Online Ensemble Using Adaptive Windowing for Data Streams with Concept Drift”. *International Journal of Distributed Sensor Networks*, 2016.
65. Sutton, Richard S and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.
66. Tsymbal, Alexey. “The problem of concept drift: definitions and related work”. *Computer Science Department, Trinity College Dublin*, 2004.

67. Tsymbal, Alexey, Mykola Pechenizkiy, Pádraig Cunningham, and Seppo Puuronen. “Dynamic integration of classifiers for handling concept drift”. *Information Fusion*, 9(1):56–68, 2008.
68. Vamplew, Peter, Richard Dazeley, Adam Berry, Rustam Issabekov, and Evan Dekker. “Empirical Evaluation Methods for Multiobjective Reinforcement Learning Algorithms”. *Machine Learning*, 84(1-2):51–80, 2011.
69. Van Moffaert, Kristof, Madalina M Drugan, and Ann Nowé. “Hypervolume-Based Multi-Objective Reinforcement Learning”. *International Conference on Evolutionary Multi-Criterion Optimization*, 352–366. Springer, 2013.
70. Van Moffaert, Kristof, Madalina M Drugan, and Ann Nowé. “Scalarized multi-objective reinforcement learning: Novel design techniques”. *IEEE Symposium on Adaptive Dynamic Programming And Reinforcement Learning (ADPRL)*, 191–199. IEEE, 2013.
71. Webber, Frederick C and Gilbert L Peterson. “Enhancing Multi-Objective Reinforcement with Concept Drift”. *Proceedings of the 30th Florida Artificial Intelligence Research Society Conference*. 2017.
72. Widmer, Gerhard and Miroslav Kubat. “Effective learning in dynamic environments by explicit context tracking”. *European Conference on Machine Learning*, 227–243. Springer, 1993.
73. Widmer, Gerhard and Miroslav Kubat. “Learning in the Presence of Concept Drift and Hidden Contexts”. *Machine Learning*, 23(1):69–101, 1996.
74. Zhou, Aimin, Yaochu Jin, Qingfu Zhang, Bernhard Sendhoff, and Edward Tsang. “Prediction-based population re-initialization for evolutionary dynamic multi-objective optimization”. *Evolutionary Multi-Criterion Optimization*, 832–846. Springer, 2007.
75. Žliobaitė, Indrė. *Learning Under Concept Drift: an Overview*. Technical report, Vilnius University, 2009.
76. Žliobaitė, Indrė, Mykola Pechenizkiy, and João Gama. “An overview of concept drift applications”. *Big Data Analysis: New Algorithms for a New Society*, 91–114. Springer, 2016.

Appendix A. *CDMORL via Error Signals*

This appendix presents the first approach to characterizing concept drift. It is described more thoroughly in [71] and based in large part on [60, 69]. This algorithm, CDMORL-es (for error signal), integrates the CD-RL non-stationary reinforcement learning algorithm from Section 2.6.4.3 with MOQL.

Reinforcement learning agents interact with their environment to optimize behavior. In the course of this interaction, they develop some sort of environmental model, such as a Q table. As long as that environmental model is accurate, the agent’s understanding of the environment eventually converges, and it is able to perform optimally [36]. However, as the environment shifts, the model begins to appear error prone. This method of *CDMORL* maintains an error signal designed to quantify this error.

The *CDMORL-es* algorithm transforms the residual of the received reward and the expected reward into an error signal E to characterize, detect change, and recognize concepts based on thresholds. Algorithms 9, 10, and 11 describe *CDMORL-es*. A concept model m consists of a Q^m table, the Q^m -specific policy π^{Q^m} , and a concept error signal, E_m . The model assumes that no a-priori transition information is known, though it knows the number of states, the number of actions, and the number of objectives. The error signal tracks residuals between predicted and received rewards. By comparing the error signal against a target, non-stationarity may be detected. The agent begins with one such model, which is its current model m_c , and the agent adds new models to handle previously unseen situations.

Algorithm 9 shows the MORL portion of the agent with concept drift modification. Line 9 is where concept characterization, change detection, and recognition occur. Line 16 is where the policy learn rate for management of the plasticity-stability dilemma occurs. These two functions are detailed in Algorithms 10 and 11.

The following sub-sections present how the agent’s CDL characterizes concepts, detects drift, and recognizes concepts. Then, it explains how the MORL algorithm leverages the information from the CDL: specifically, how Win or Learn Fast (WoLF)

Table 21: Parameters and their nominal values. Parameters with * can be decayed.

Parameters used in both algorithms		
Symbol	Default	Meaning
α^*	0.1	Learn rate
γ	0.9	Discount factor
δ_l^*	0.2	Policy learning rate while ‘losing’
δ_w^*	0.05	Policy learning rate while ‘winning’
ϵ^*	0.3	Probability of random action selection
Used in <i>CDMORL-es</i>		
E_{min}	0.8	Threshold for a <i>warning</i>
E_{match}	0.2	Threshold for <i>drift</i> and declaring a match
ρ	0.2	Weight of new values in the error equation
$t_{warm\ up}$	150	Time for a concept to stabilize after creation before detecting a concept change
$t_{cool\ down}$	25	Time to disallow concept change detection to prevent thrashing

Algorithm 9 The combined CDMORL-es

```

1:  $m_{cur} \leftarrow newmodel()$ 
2:  $M \leftarrow m_{cur}$ 
3:  $s \leftarrow s_0$ 
4: repeat
5:   select  $a$  according to policy  $\pi_{m_{cur}}(s)$  derived from  $Q_{m_{cur}}$  given  $s$ 
6:   Take action  $a$ 
7:   observe  $s'$ 
8:   observe reward vector  $\vec{r}$ 
9:    $m_{cur} \leftarrow SelectModel(s, a, \vec{r})$ 
10:  propose  $a'$  based on action selection technique
11:  if The model wasn't changed: then
12:    Perform the Q update:
13:    for all objective  $o$  do
14:       $Q_{m_{cur}}(s, a, o) \leftarrow Q_{m_{cur}}(s, a, o) + \alpha[\vec{r}_o + \gamma Q_{m_{cur}}(s', a', o) - Q_{m_{cur}}(s, a, o)]$ 
15:    end for
16:     $\pi_{m_{cur}}(s, a) \leftarrow WinOrLearnFast - PHC(s, a, m_{cur})$ 
17:  end if
18:   $s \leftarrow s'$ 
19: until  $s$  is terminal

```

is incorporated to mitigate issues related how catastrophic forgetting is addressed and the plasticity-stability dilemma. The end of this section presents an analysis of this algorithm, including its shortcomings and a method to address these shortcomings.

A.1 Concept Characterization, Change Detection, and Recognition

Concept characterization, change detection, and recognition is centered on an error signal and two thresholds, as detailed in Algorithm 10. The concept drift learner is adapted from [59], which employs an error signal that is based, in part, on the error between the expected and actual reward. The CDL also employs scalarization in the form of indicators (see Section 2.3) and also a threshold for *warning*, similar to many other CDLs (see Section 2.5).

Algorithm 10 SelectModel

Require: $s, a, \vec{r}$

```

1: if  $t_{\text{since concept creation}} < t_{\text{warm up}}$  or  $t_{\text{since concept change}} < t_{\text{cool down}}$  then
2:   return  $m_{\text{cur}}$ 
3: end if
4: for all  $m \in M$  do
5:    $\Delta R_m \leftarrow \mathbf{Q}(s, a) - \vec{r}$ 
6:    $e_m \leftarrow 1 - 2\text{scalarize}\left(\frac{(\Delta R_m)^2}{(R_{\text{max}} - R_{\text{min}})^2}\right)$ 
7:    $E_m \leftarrow E_m + \rho(e_m - E_m)$ 
8: end for
9: if  $E_{m_{\text{cur}}} < E_{\text{match}}$  then
10:   $m_{\text{cur}} \leftarrow \text{argmax}_m(E_m)$ 
11: end if
12: if  $E_{m_{\text{cur}}} < E_{\text{match}}$  then
13:   $m_{\text{cur}} \leftarrow \text{newmodel}()$ 
14:   $M \leftarrow M \cup m_{\text{cur}}$ 
15: end if
16: return  $m_{\text{cur}}$ 

```

All concept models m in the ensemble, whether active or not, maintain a running signal E_m , which is increased or decreased based on a weight ρ and the instantaneous error. A model's instantaneous error e_m is a function of the residual between the expected utility $\vec{q}_m$ and the observed reward, $\vec{r}$, as shown in Equation 15. The instantaneous error provides a signal where 1 indicates a perfectly accurate utility function and -1 signifies a completely wrong function.

$$e_m = 1 - 2\frac{(\Delta R_m)^2}{(R_{\text{max}} - R_{\text{min}})^2}. \quad (15)$$

However, this only works for a single objective, and requires the agent to have a model of $R(s, a)$ (because ΔR_m is the difference between the expected and received reward, that is, $\Delta R_m = R(s, a) - r(t)$). An indicator is used to convert the multiple objectives into a scalar value. This research uses the hypervolume indicator [12, 69], because it is Pareto-compliant. To remove the requirement on learning $R(s, a)$, the agent instead uses its utility $Q(s, a)$ values¹. The new instantaneous error is given in Equation 16 (The denominator is an element-wise square and then element-wise division.).

$$e_m = 1 - 2 \text{scalarize} \left(\frac{(\Delta R_m)^2}{(R_{max} - R_{min})^2} \right) \quad (16)$$

As in [59], the instantaneous error e is added back into the running error E by weighting it by a factor of ρ . However, in [59], the running error is a function of both an instantaneous error for the reward and the instantaneous error for the transition. Because this work aims to be model-free, no transition function is assumed, and to maintain generality and be better able to support continuous environments, no transition function is learned. Accordingly, the weight Ω between the residual reward and the transition residual can be set to zero out the transition instantaneous error, and focus only on the instantaneous error for the reward. For simplicity, the confidence $c_m(s, a)$ of the state, action pair for the given model is be set to 1. This eliminates the overhead of the tracking required and prevents the coefficient from approaching 0 as the number of states increase and as the memory increases². The resulting equation for running error becomes

$$E_m = E_m + \rho(e_m - E_m). \quad (17)$$

¹This is a flawed idea, because Q is a utility and not the instantaneous reward, except when γ is 0.

²As will be mentioned later, it is important to track visits to state-action pairs in some way to prevent over-relying on new state-action pairs for drift characterization. This is discussed further in the next algorithm.

While the running error for the current model E_{m_c} remains above E_{min} , the CDL characterizes the concept as *stable*, so learning is slowed and when E_{m_c} drops below E_{min} , the concept characterization is *warning*. When E_{m_c} drops below E_{match} , concept *drift* is detected.

Because of the simplification removing the confidence $c_m(s, a)$ weight, it is necessary to ensure that a concept has sufficient time to converge. Previous analysis assessed that it takes on the order of $O(|S||A|(\log(1/\epsilon)/\epsilon^2)(\log(N/\delta) + \log \log(1/\epsilon)))$ to converge with probability $1 - \delta$ to within ϵ of the optimal policy [36] (note that this is a different usage of ϵ than in the rest of this dissertation). Because the number of visits gets very large to ensure reasonable performance, this method instead insists on having a minimum number of steps to converge, $t_{warm\ up}$.

A.2 Integrating the CDL into MORL

The information from the CDL provides the MORL agent with the information it needs to mitigate catastrophic forgetting, as shown in lines 9-15 of Algorithm 10, and the plasticity-stability dilemma, as shown in Algorithm 11.

A.2.1 Mitigating Catastrophic Forgetting. The agent mitigates catastrophic forgetting as follows. First, when the CDL detects drift (that is, the current concept goes below E_{match}), it stores the current concept. This means that the Q table and policy π are stored, as is the running error E . Each step, the stored policy will update E as though it was active, but it won't impact the learning rate. When the current model detects drift, it checks its previously stored policies for the concept with the greatest error score above E_{match} . That concept is reloaded. If there are no concepts above E_{match} , then a new concept is created.

A.2.2 Mitigating the Plasticity-Stability Dilemma. To address the plasticity-stability dilemma, the algorithm uses a variable learn rate, as in Win or Learn Fast [6], shown in Algorithm 11. *CDMORL-es* follows the strategy of WoLF by using a notion

of ‘winning’ and ‘losing,’ which both only impact the probabilistic policy $\pi(s, a)$. An agent is deemed ‘winning’ when the algorithm is stable. When a *warning* is detected by the CDL, the algorithm uses the ‘losing’ learn rate (which is greater than the ‘winning’ rate). Algorithm 11 shows the modified Win or Learn Fast - Policy Hill Climbing (PHC) algorithm.

Algorithm 11 WinOrLearnFast-PHC

Require: s, a, m_{cur}

- 1: $\delta_l > \delta_w \in (0, 1]$
- 2: **if** $E_{m_{cur}} > E_{min}$ **then**
- 3: $\delta \leftarrow \delta_w$
- 4: **else**
- 5: $\delta \leftarrow \delta_l$
- 6: **end if**
- 7: $\delta_{sa} = \min \left(\pi_{m_{cur}}(s, a), \frac{\delta}{|A_{legal}| - 1} \right)$
- 8: **if** $a \neq \operatorname{argmax}_{a'} Q_{m_{cur}}(s, a')$ **then**
- 9: $\Delta_{s,a} \leftarrow -\delta_{sa}$
- 10: **else**
- 11: $\Delta_{s,a} = \sum_{a' \neq a} \delta_{sa'}$
- 12: **end if**
- 13: $\pi_{m_{cur}}(s, a) \leftarrow \pi_{m_{cur}}(s, a) + \Delta_{s,a}$
- 14: **return** $\pi_{m_{cur}}(s, a)$

A.3 Analysis

The design of CDMORL-es requires a reasonable estimate of $R(s, a)$ before it attempts to detect drift. The algorithm that CDMORL-es is modeled after, CDRL [60], is based on the residual between $\mathbf{R}(s, a)$ and $\vec{r}$. However, the $Q(s, a)$ is a utility function, so its value is adjusted based on future expected rewards. This means there is always a small amount of error. It’s worth noting that [60] scales the instantaneous error signal based on the number of visits to a specific state and action. However, it is always a small coefficient, which takes many steps before the impact is felt. Instead of using a small coefficient, *CDMORL-es* uses a freeze time which allows the current concept a chance to converge. Both allow unvisited concepts to potentially cause an assessment of *drift*.

CDMORL-es mitigates the plasticity-stability dilemma by managing the learn rate based on the stability of the concept: *stable* concepts receive a lesser learn rate, to prevent noise from corrupting a reliable definition, while *warning* concepts receive a greater learn rate to improve convergence. The reverse was experimentally attempted, which appeared to result in noise being added to otherwise stable concepts while convergence in unvisited states was diminished.

The inability of *CDMORL-es* to demonstrate a consistent and compelling improvement prompted additional research into concept definition in relationship to characterization, into recognition practices, and into noise mitigation. The findings, as discussed in Chapter II, were that concept definition should not occur as a concept has drifted, but when it was stable; that recognition is typically performed in numerical domains using statistical measures; and batches to mitigate noise. The method in Chapter III presents a form of Concept Drift-aware MORL (*CDMORL*) that incorporates these findings. The limitations of *CDMORL-es* addressed are:

1. Unvisited state-action pairs are able to affect the concept, making it possible that a correct concept will receive significant adverse feedback or that a wrong concept will receive significant positive feedback.
2. This method fails to capitalize on the multi-objective nature of the environment because it scalarizes the residual before checking for drift, rather than afterward.
3. Concept recognition can only happen at the same time step that *drift* is detected. *CDMORL-es* and many other algorithms only check at the moment drift is detected, though the opposite is that RL-CD checks at every time step [60]).
4. Parameter sensitivity: the error signal for *CDMORL-es* is heavily dependent on two sensitive thresholds, error absorption rate ρ , a sensitive initialization time $t_{warm\ up}$, an anti-thrash time $t_{cool\ down}$, a substantial dependency on how the Q table was designed and initialized, and dependent on knowledge of the maximum and minimum possible values for each reward.

Due to these limitations, which resulted in very poor results demonstrating at best a weak improvement, a new algorithm was designed. CDMORL as documented in Chapter III incorporates several complete algorithmic changes, maintaining only the high level philosophy of the previous algorithm: it is possible to characterize, detect drift, and recognize concept drift by computing the residual between the observed reward $\vec{r}$ and the estimated utility for the current concept $\vec{q}_c$. The second algorithm improves on several shortcomings in the first algorithm. The two key improvements are the switch to characterizing concept drift using state and action specific multi-variate statistics away from the scalarized state and action-independent error signal; and an overhaul of the recognition mechanism to recognize previously stable concepts when the current concept is characterized as drifted or initializing, as opposed to only recognizing previously unstable concepts at the moment the concept is characterized as drifted.

In summary, this develops concept drift-aware multi-objective reinforcement learning using an error signal based on an indicator scalarization of the residual between the expected utility and the actual reward. This algorithm is discussed in [71], demonstrating a minimal return for the additional computation.

Appendix B. CDMORL via Statistics

This appendix discusses a statistical approach to concept drift classification. Ultimately, this was deemed inappropriate for problems where the reward signal is deterministic, as this causes two problems: without noise, the standard deviation in a statistical approach will collapse to zero, causing numerical instabilities; and without a variety of rewards (whether from a non-deterministic transition or because a state-action pair has multiple modes), the residual will continuously converge, creating a continuously creeping mean, which makes classification challenging because each new sample is near the outer edge of the distribution. The rest of this appendix describes this approach, which is still promising for future study in non-deterministic noisy environments.

This version maintains a statistical model of the residual between the utility vector and the observed reward vector each state-action pair and computes the probability of inclusion using Student’s t-distribution [43] to determine the probability of a sample being excluded from the current concept.

Each concept maintains a window of samples $W_{s,a}$ of the residual between the expected utility for the concept $\vec{q}_c$ and observed reward $\vec{r}$, that contains between zero and N_W samples (discarding oldest). When a sample is classified, if that is the first or second sample, the window is still *initializing*. Starting with the third sample, the sample classifier design is modeled after the Early Drift Detection Method (EDDM) [2]. For each sample, the CDL computes, for each concept c and each objective o , a probability $p_{i,o}(\vec{q} - \vec{r}|s, a)$ via Student’s t-test, where i indicates how many samples are in the concept’s window (so, the third sample for a concept is $i = 3$).

The probability that the sample is part of the distribution is computed using the t-statistic and the degrees of freedom for each objective o , yielding $p_{o,i}$ (the probability that a value is more extreme within the distribution), which is then converted using $p'_{o,i} = 1 - p_{o,i}$. By assuming the objectives are independent, this test can be conducted for each objective o , and the product of each individual probability is the probability

for the sample $p'_i = \prod_o(p'_{o,i})$. The standard deviation is computed per [22]: $s'_i = \sqrt{p'_i(1 - p'_i)/i}$. To prevent numerical instabilities, the s values are not permitted to go below 0.001. The p'_{max} and s'_{max} values are updated when $p'_{max} + 2s'_{max} > p'_i + 2s'_i$. These are reset for new concepts.

Table 22: Parameters and their nominal values. Parameters with * can be decayed.

Symbol	Default	Meaning
β_1	0.9	Threshold for a <i>warning</i>
β_2	0.8	Threshold for <i>drift</i>
N_W	11	Window size
N_B	3	Batch size

Once the ratio $(p'_i + 2 \cdot s'_i)/(p'_{max} + 2 \cdot s'_{max})$ falls below a threshold β , the sample is declared to indicate either a warning or drift. The ratios used here are that ratios below β_2 indicate *drift* and between β_2 and β_1 indicate a *warning*, that drift might be happening. Algorithm 8 describes the sample classification process.

After checking for drift, the algorithm is careful to remove samples from all algorithms after concepts are checked. After checking concepts and responding to *drift* detections, the active concept is guaranteed to not be *drifted*. Thus, learning occurs at the end of each step.

Algorithm 12 Characterize Sample

```
1: function CHARACTERIZE SAMPLE( $s, a, \vec{q}_c(s, a), \vec{r}$ )
2:    $residual \leftarrow \vec{q}_{c,s,a} - \vec{r}$ 
3:   if residual history of  $s, a$  is shorter than two then
4:     add  $residual$  to history for  $s, a$ 
5:      $label \leftarrow init$ 
6:   else
7:     Compute  $p_i, s_i$  from  $residual$  history of  $s, a$ 
8:      $ratio \leftarrow (p_i + 2s_i)/(p_{best} + 2s_{best})$ 
9:     if  $ratio < \beta$  then
10:       $label \leftarrow drift$ 
11:     else if  $ratio < \alpha$  then
12:       $label \leftarrow warn$ 
13:     else
14:       $label \leftarrow stable$ 
15:     end if
16:     if  $p_i + 2s_i > p_{max} + 2s_{max}$  then
17:       Update  $p_{max}$  and  $s_{max}$ 
18:     end if
19:   end if
20:   return  $label$ 
21: end function
```

Appendix C. Comparison of Subsets of MERLIN Results

This appendix provides the remainder of the data for MERLIN in the stationary environment.

C.1 Comparison of non-Batch Agents

Figures 43-48 show the performance of batch-based CDMORL on the six non-stationary instantiations of MERLIN.

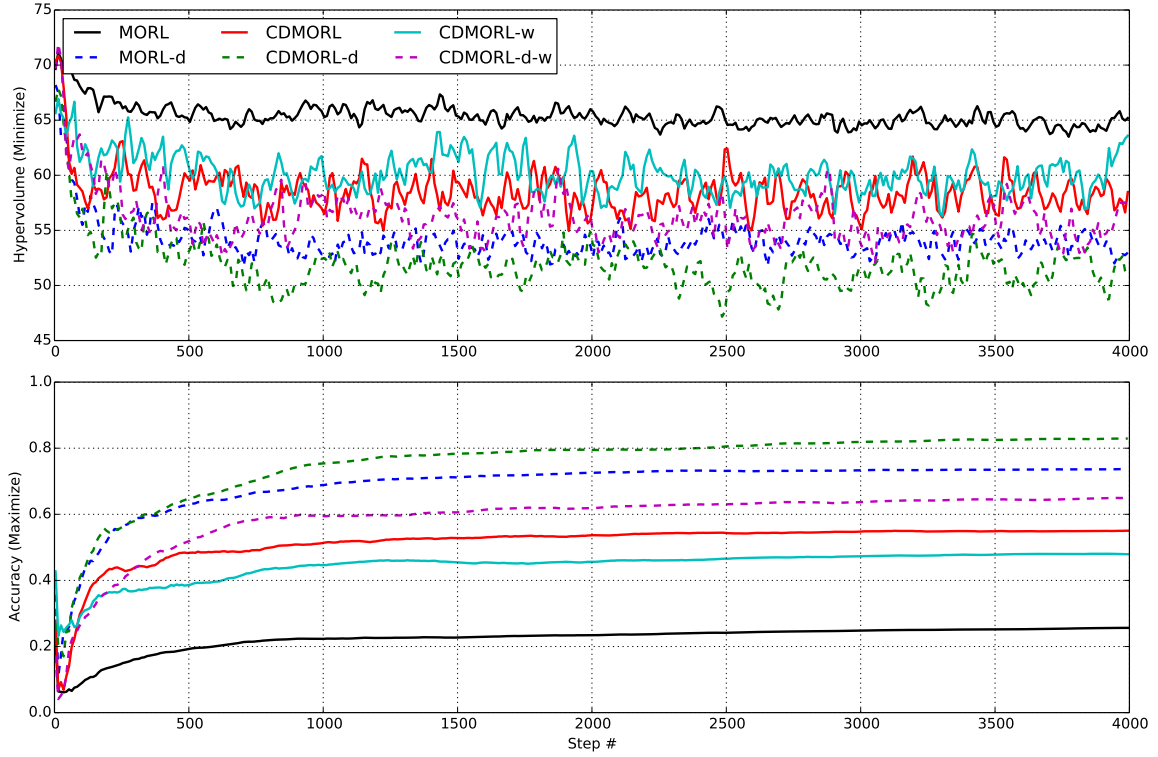


Figure 43: Hypervolume and accuracy for drifting MERLIN experiment 1, comparison among non-Batch agents and reference agents.

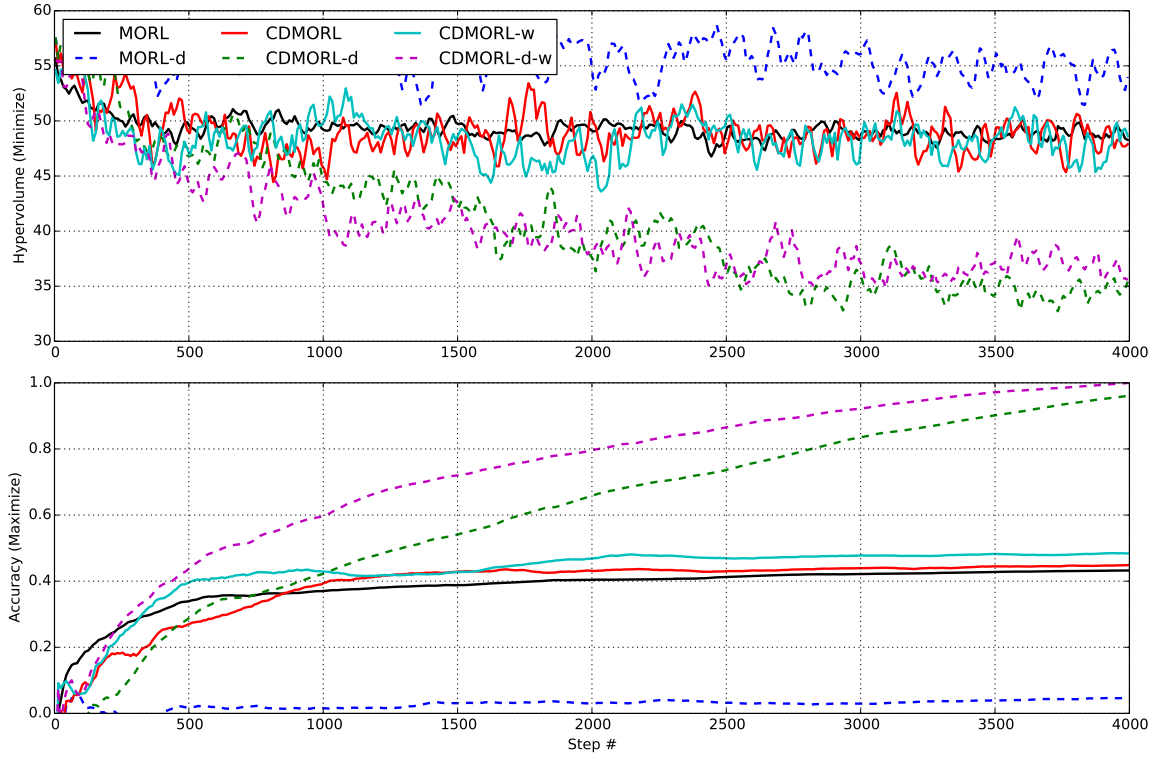


Figure 44: Hypervolume and accuracy for drifting MERLIN experiment 2, comparison among non-Batch agents and reference agents.

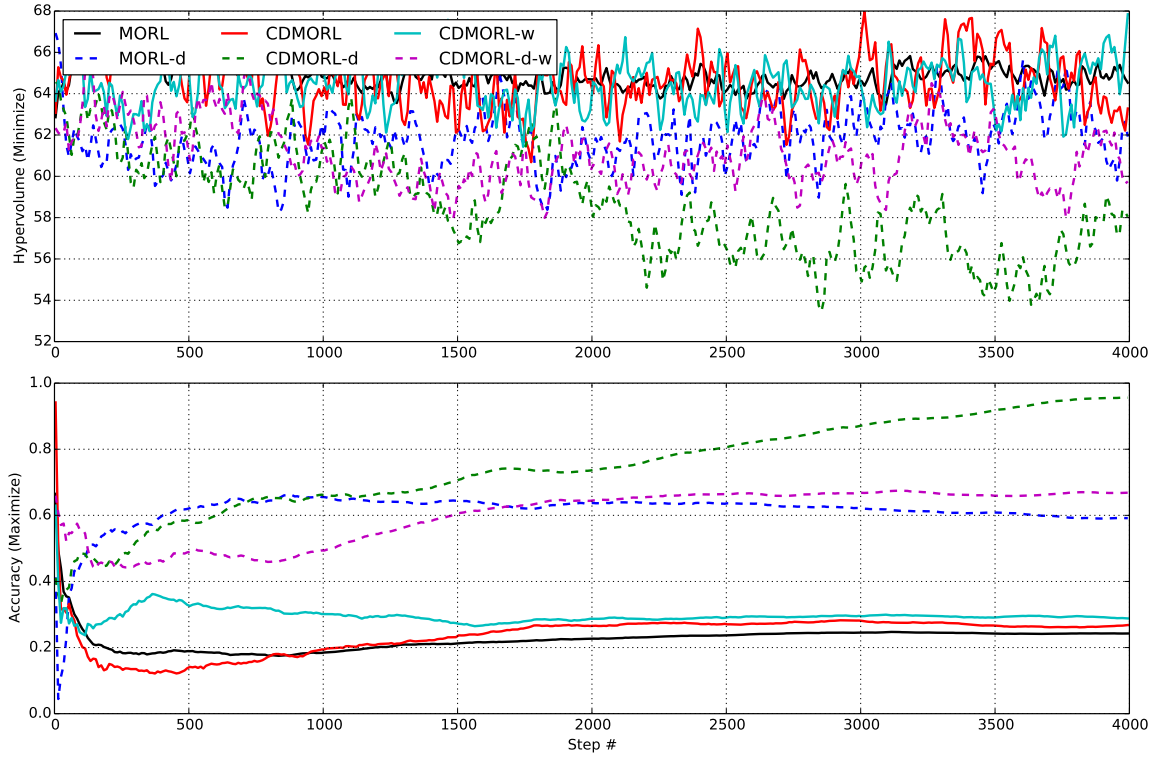


Figure 45: Hypervolume and accuracy for drifting MERLIN experiment 3, comparison among non-Batch agents and reference agents.

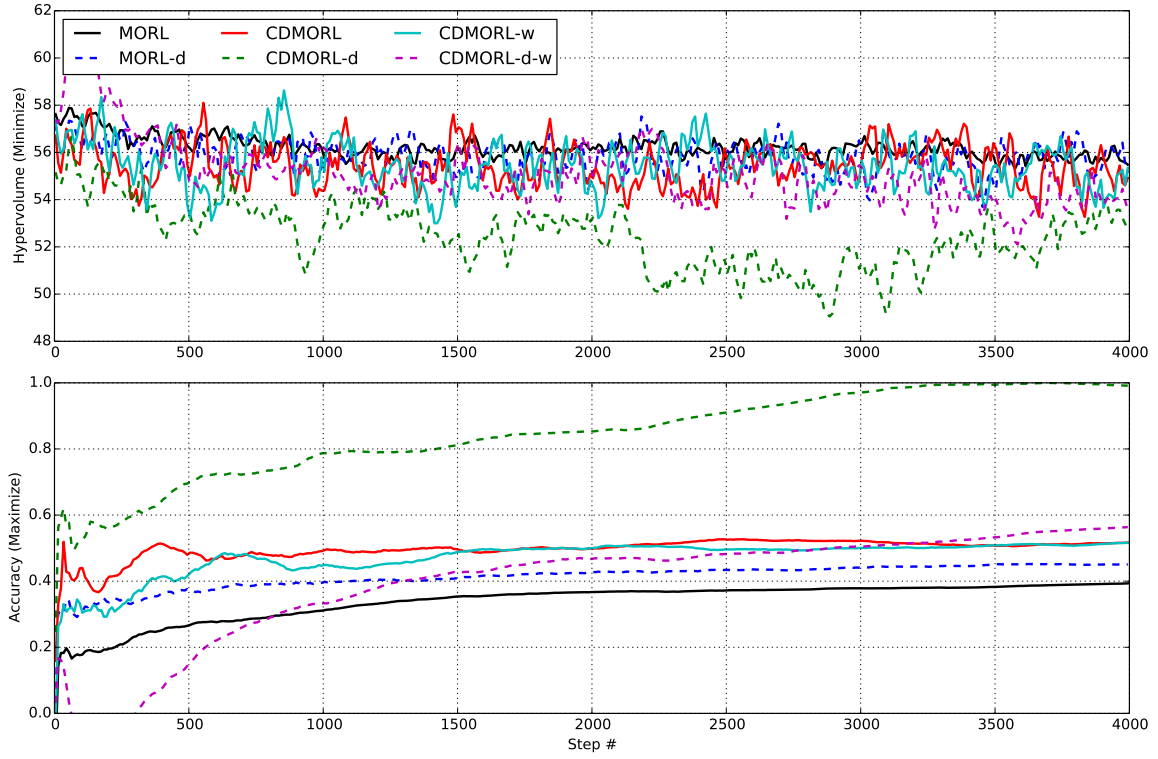


Figure 46: Hypervolume and accuracy for drifting MERLIN experiment 4, comparison among non-Batch agents and reference agents.

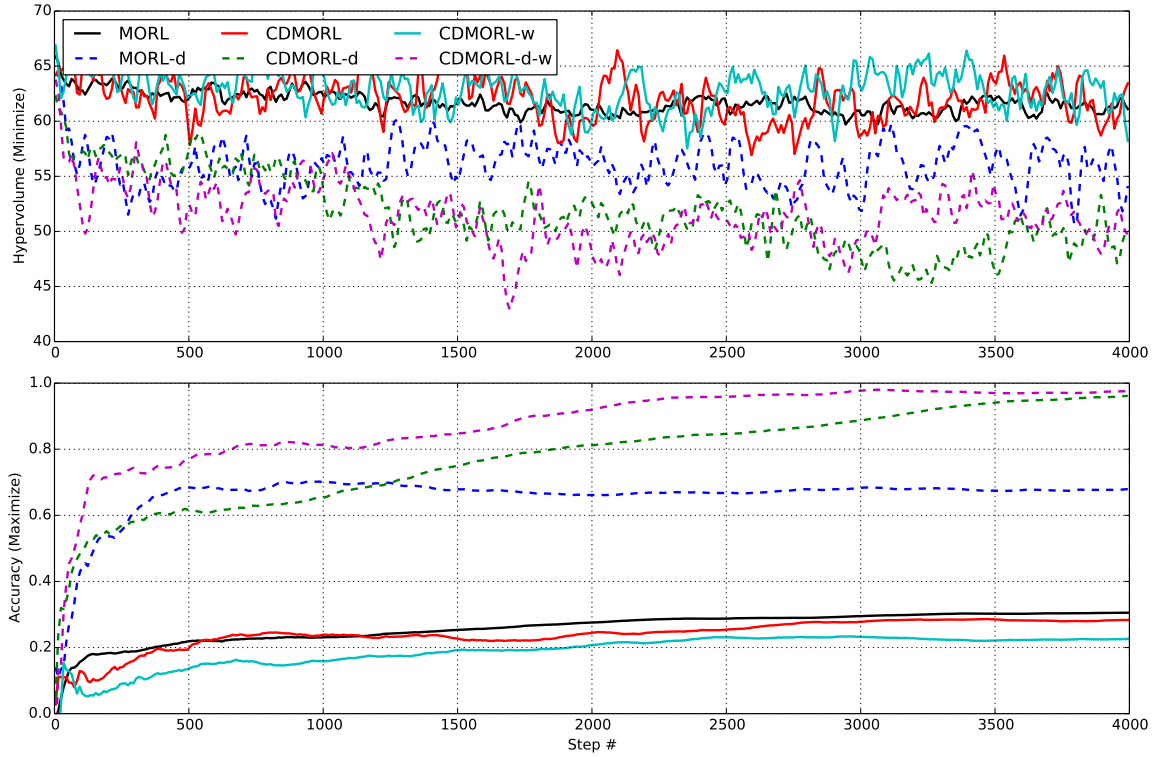


Figure 47: Hypervolume and accuracy for drifting MERLIN experiment 5, comparison among non-Batch agents and reference agents.

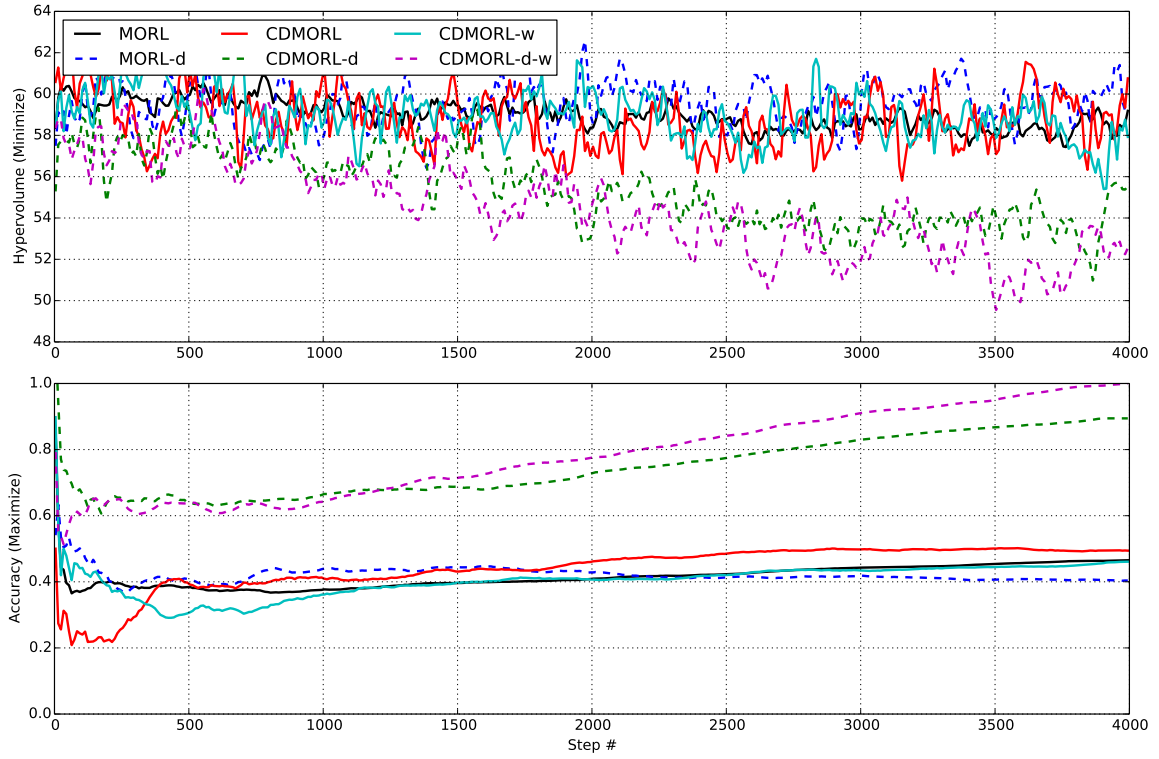


Figure 48: Hypervolume and accuracy for drifting MERLIN experiment 6, comparison among non-Batch agents and reference agents.

C.2 Comparison of Batch Agents

The performance of CDMORL with batches is shown in Figures 49 - 54.

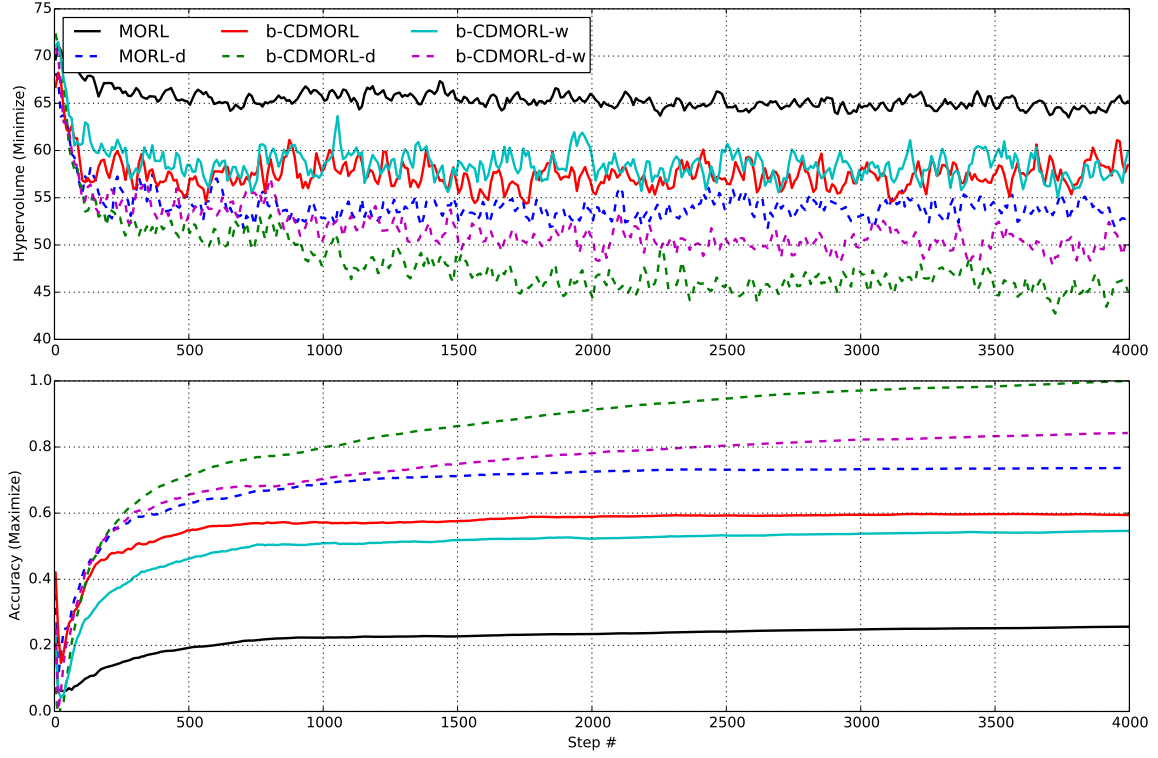


Figure 49: Hypervolume and accuracy for drifting MERLIN experiment 1, comparison among Batch agents and reference agents.

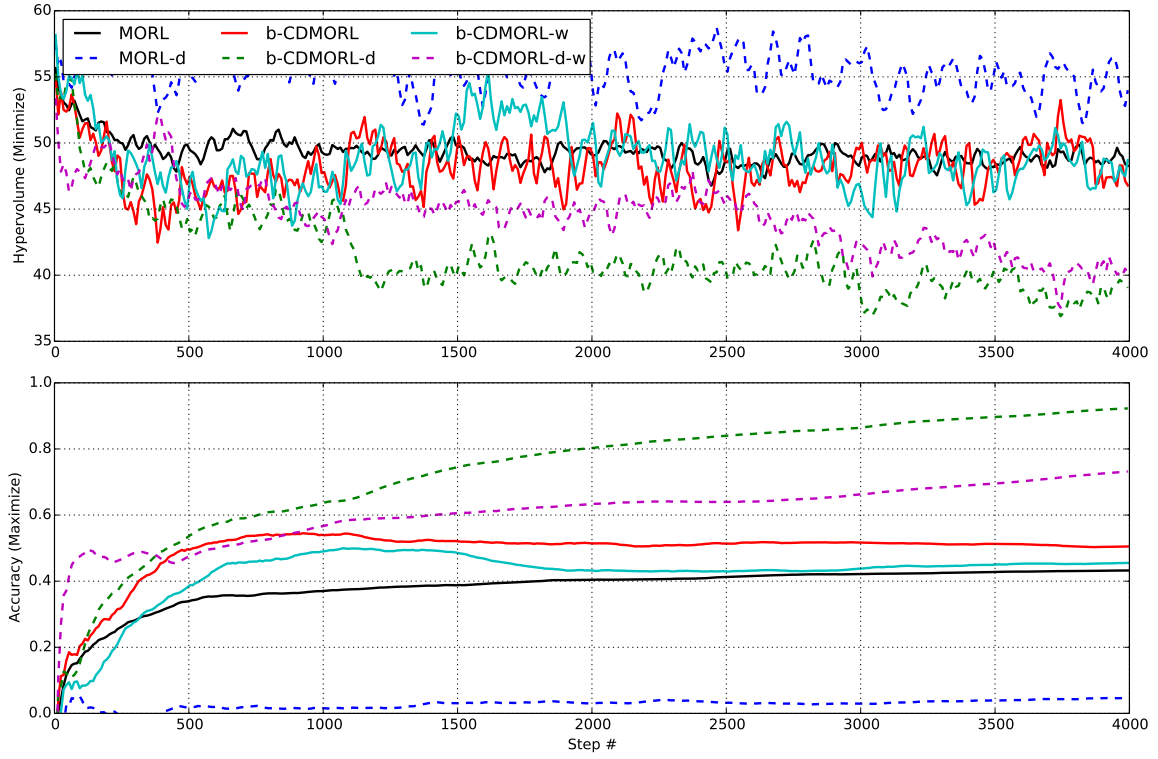


Figure 50: Hypervolume and accuracy for drifting MERLIN experiment 2, comparison among Batch agents and reference agents.

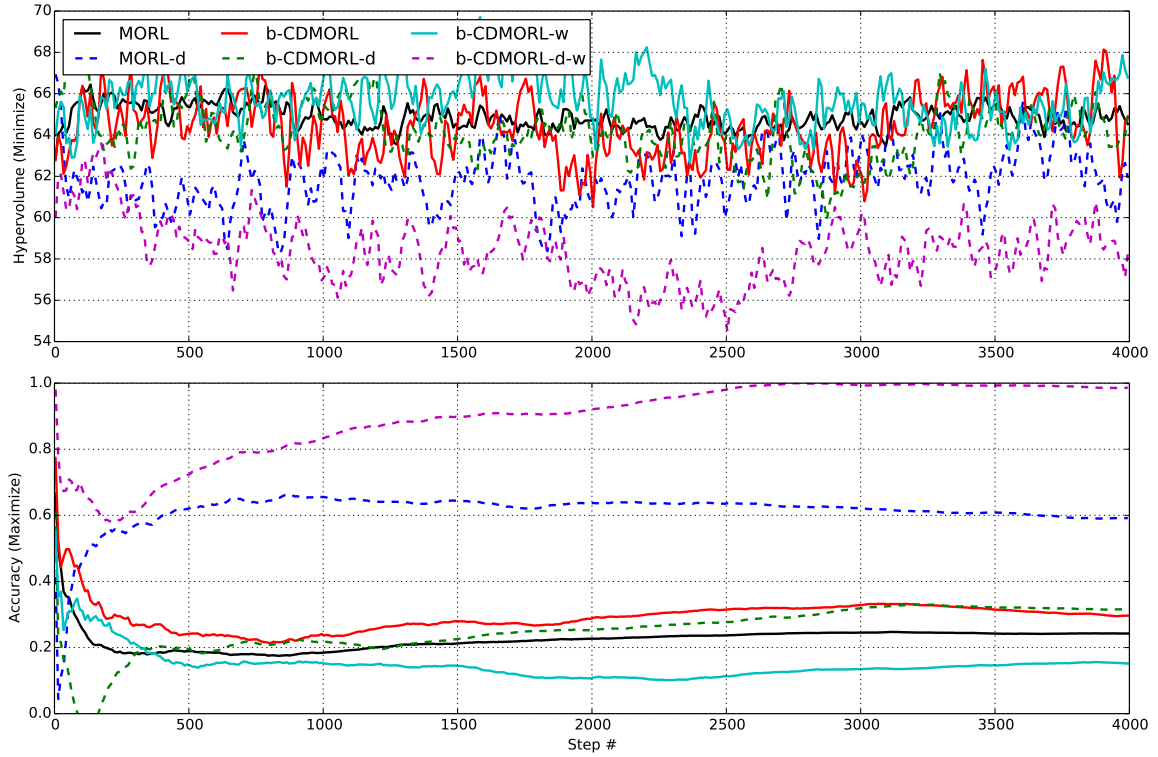


Figure 51: Hypervolume and accuracy for drifting MERLIN experiment 3, comparison among Batch agents and reference agents.

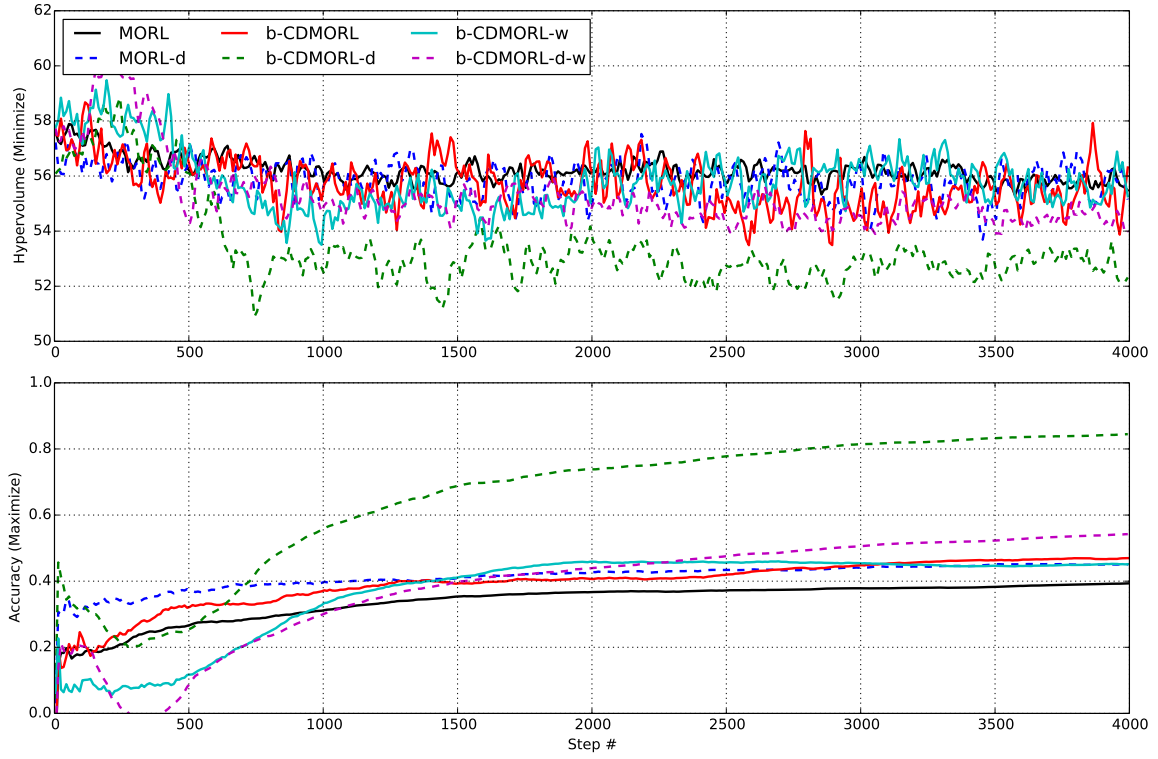


Figure 52: Hypervolume and accuracy for drifting MERLIN experiment 4, comparison among Batch agents and reference agents.

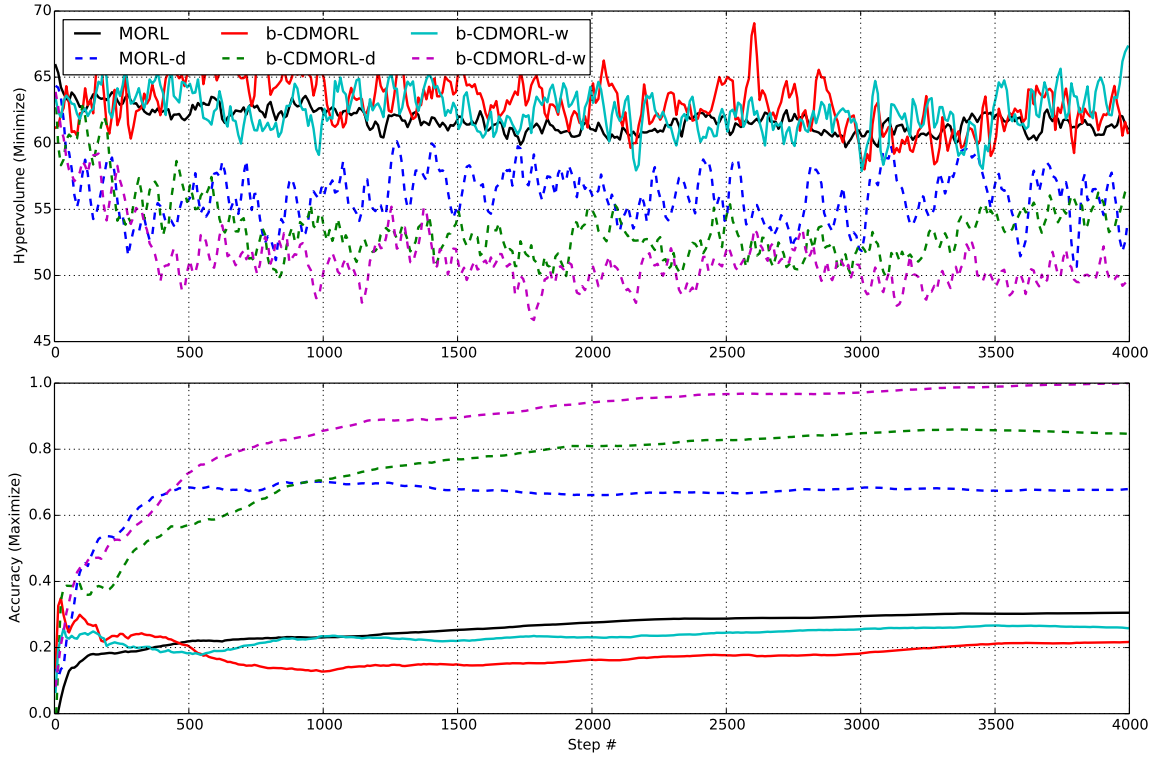


Figure 53: Hypervolume and accuracy for drifting MERLIN experiment 5, comparison among Batch agents and reference agents.

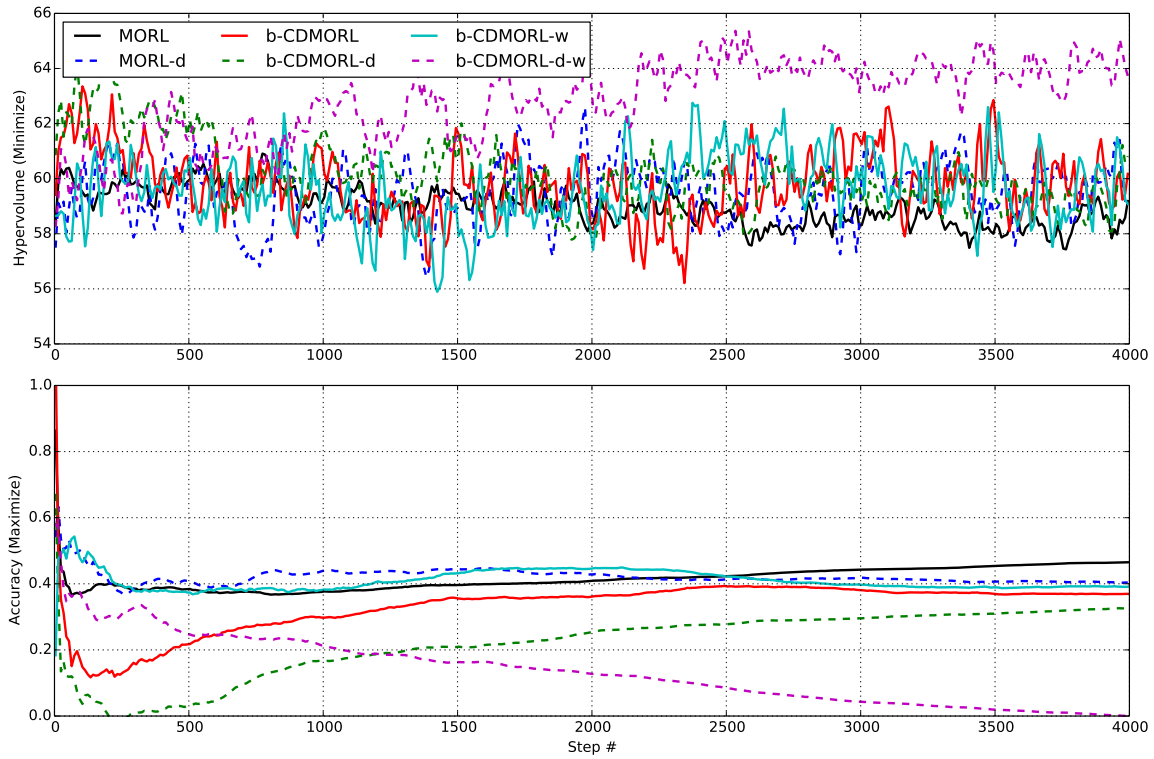


Figure 54: Hypervolume and accuracy for drifting MERLIN experiment 6, comparison among Batch agents and reference agents.

C.3 Comparison of Single Objective Agents

This section presents agents using only a single objective (SO) for optimizing.

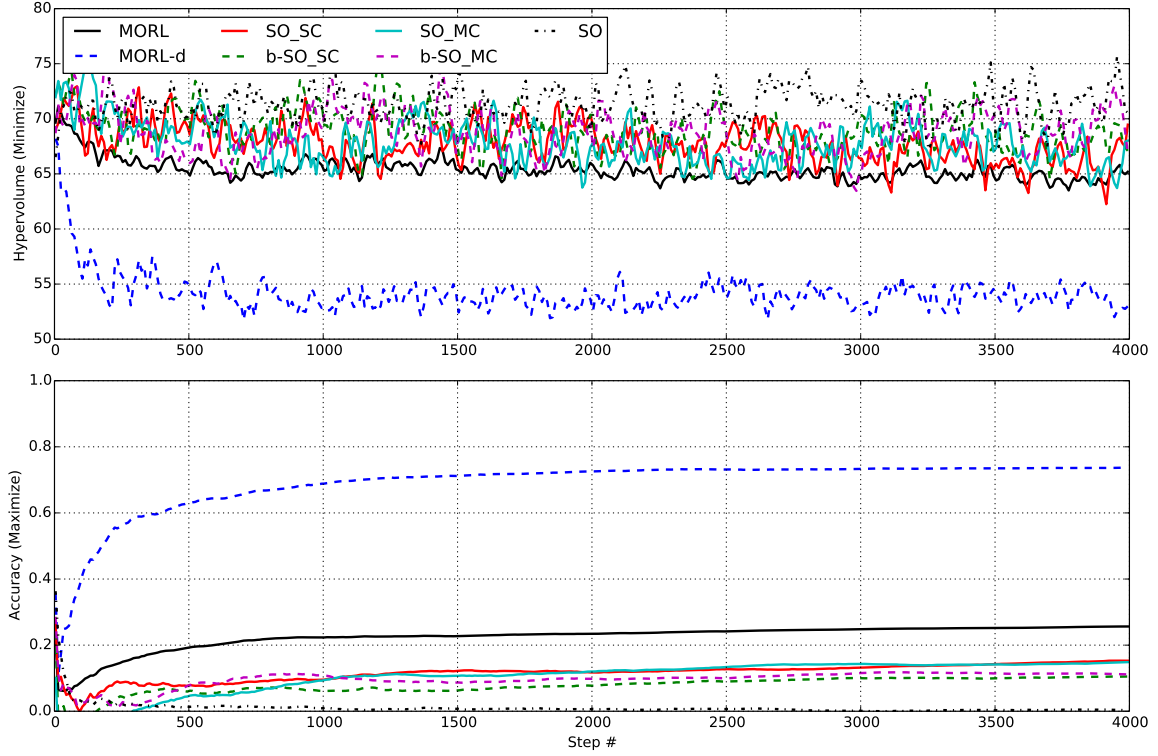


Figure 55: Hypervolume and accuracy for drifting MERLIN experiment 1, comparison among single objective agents and reference agents.

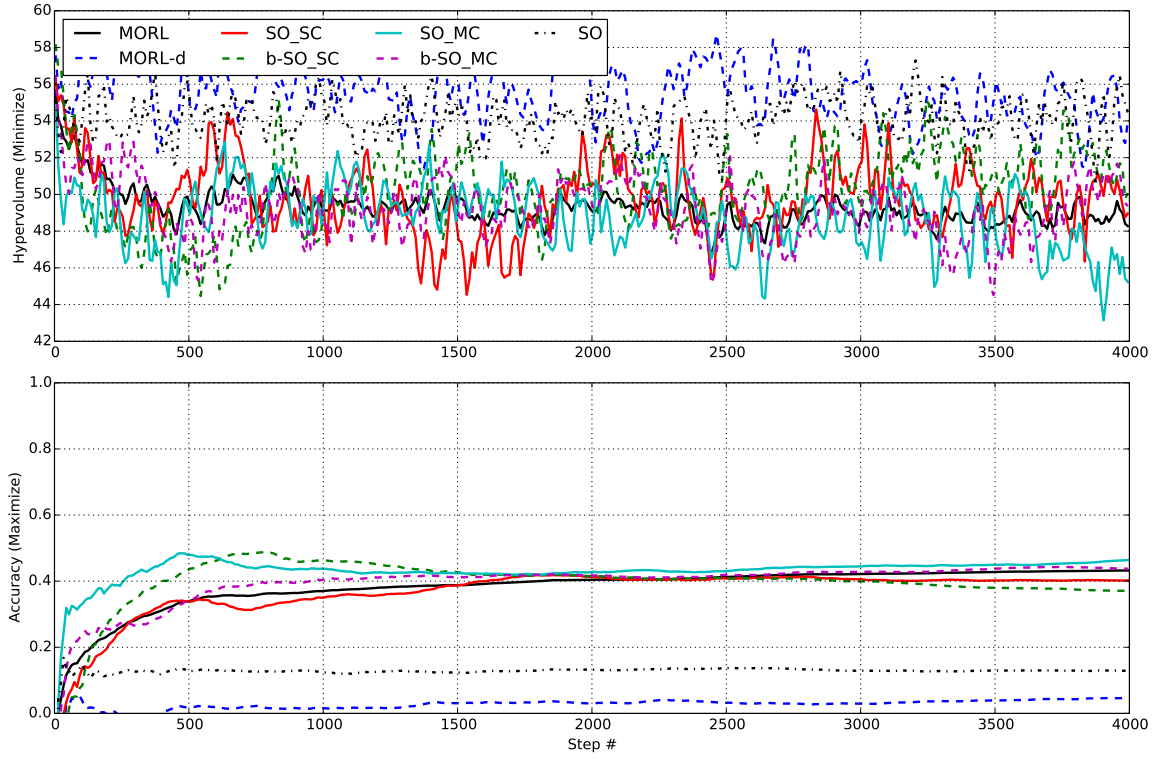


Figure 56: Hypervolume and accuracy for drifting MERLIN experiment 2, comparison among single objective agents and reference agents.

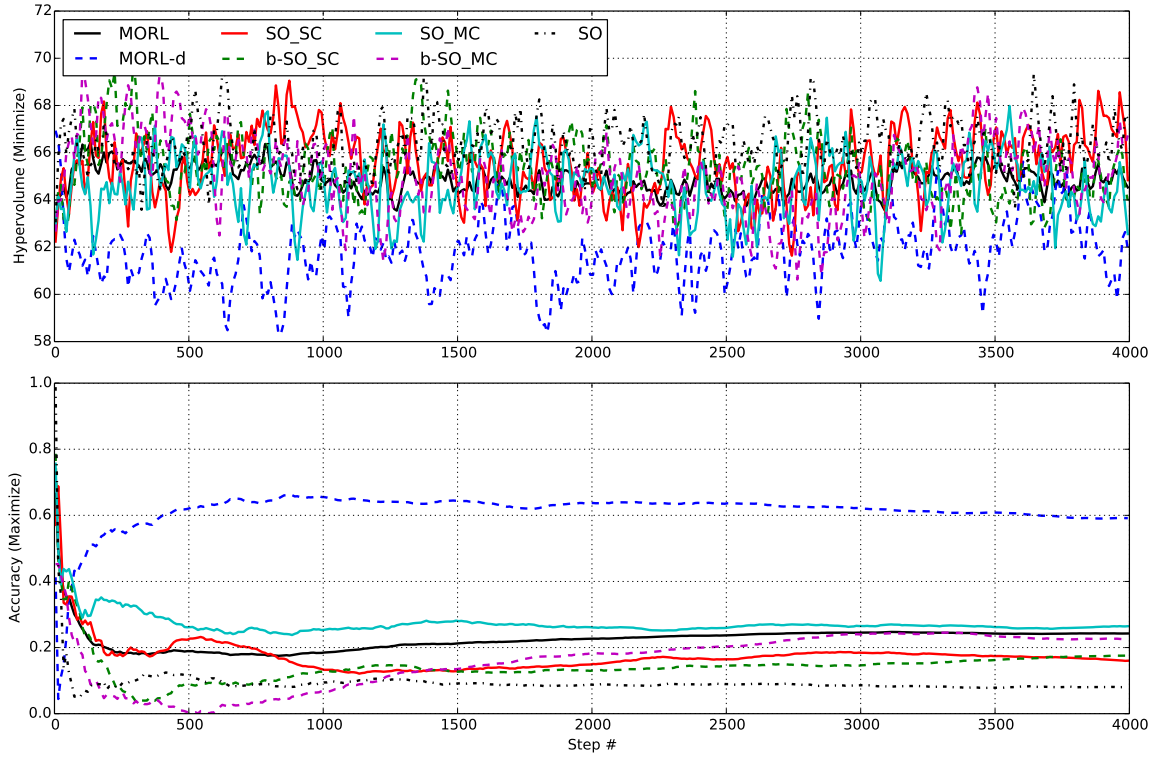


Figure 57: Hypervolume and accuracy for drifting MERLIN experiment 3, comparison among single objective agents and reference agents.

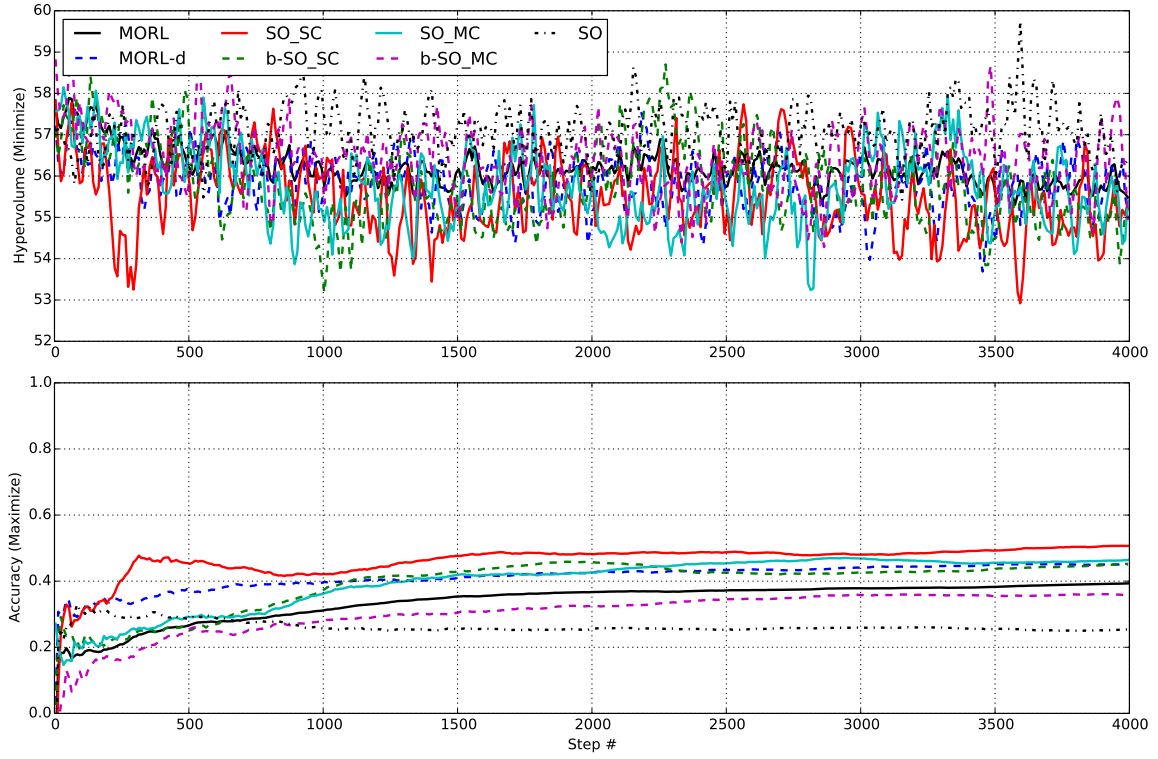


Figure 58: Hypervolume and accuracy for drifting MERLIN experiment 4, comparison among single objective agents and reference agents.

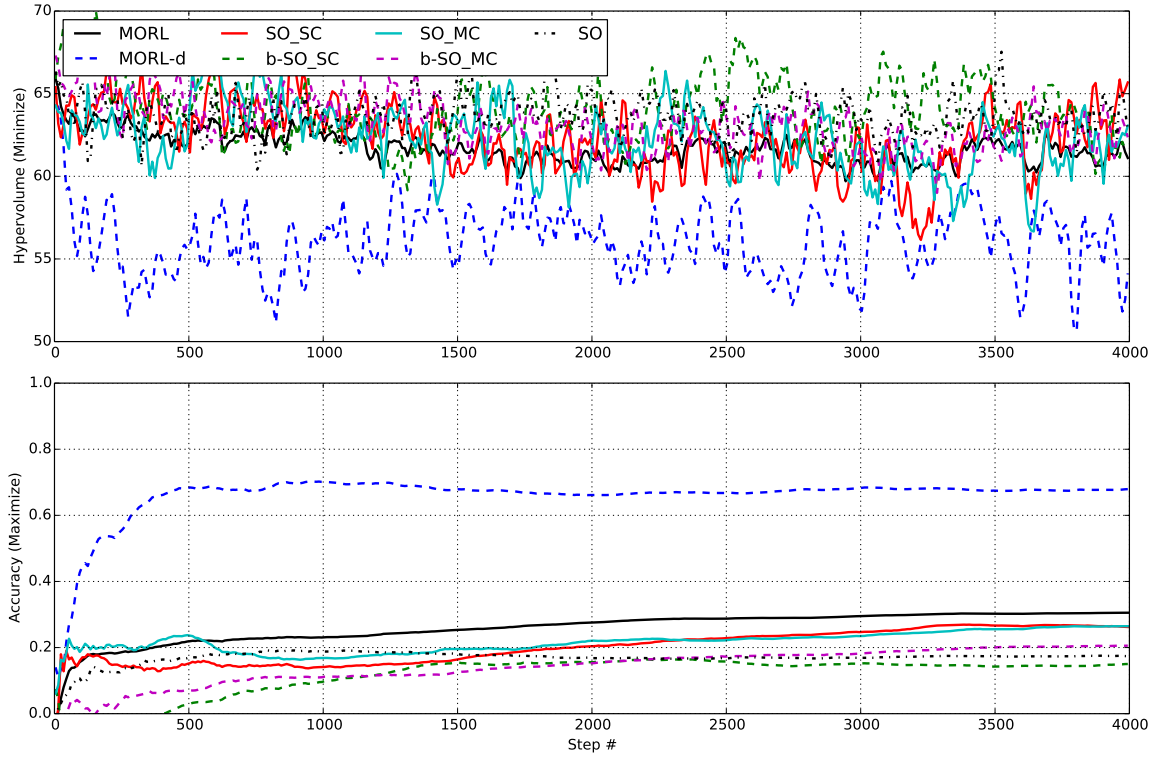


Figure 59: Hypervolume and accuracy for drifting MERLIN experiment 5, comparison among single objective agents and reference agents.

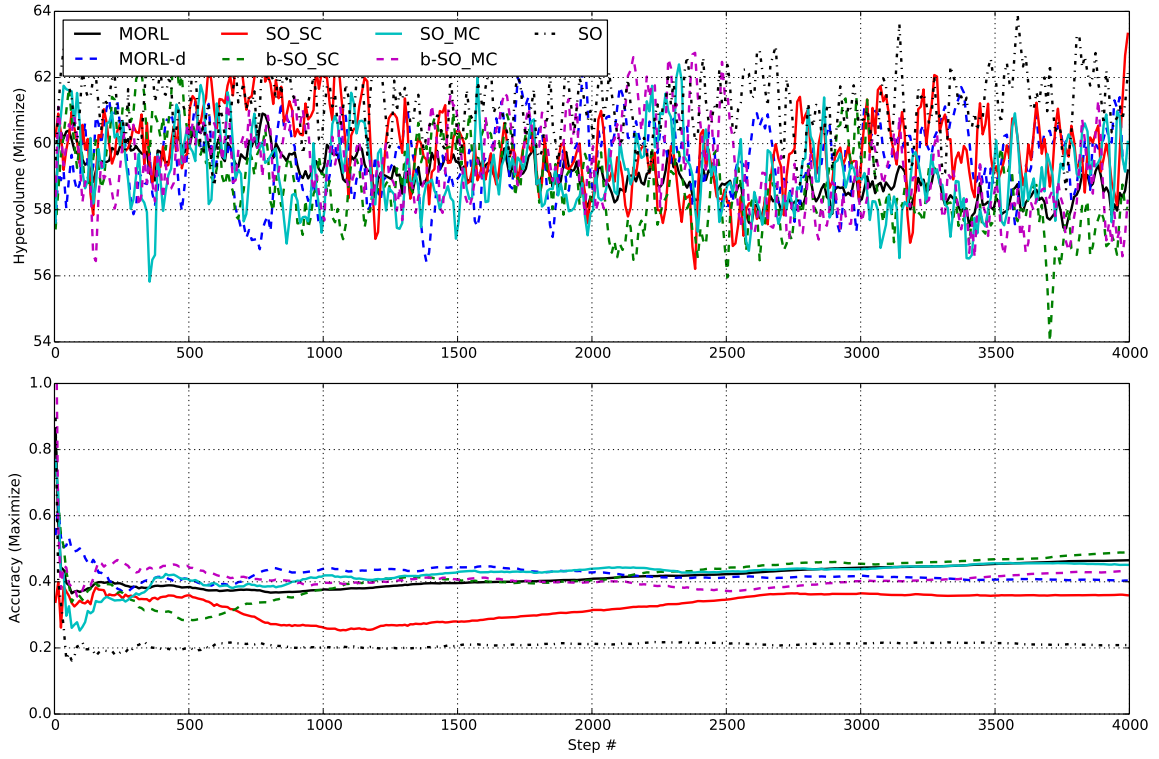


Figure 60: Hypervolume and accuracy for drifting MERLIN experiment 6, comparison among single objective agents and reference agents.

REPORT DOCUMENTATION PAGE					<i>Form Approved</i> OMB No. 0704-0188	
The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.						
1. REPORT DATE (DD-MM-YYYY) 21-12-2017		2. REPORT TYPE Dissertation			3. DATES COVERED (From — To) Sept 2009 — Dec 2017	
4. TITLE AND SUBTITLE Multi-Objective Reinforcement Learning with Concept Drift					5a. CONTRACT NUMBER 5b. GRANT NUMBER n/a 5c. PROGRAM ELEMENT NUMBER	
6. AUTHOR(S) Webber, Frederick, C, Civilian					5d. PROJECT NUMBER 5e. TASK NUMBER 5f. WORK UNIT NUMBER	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 Hobson Way WPAFB OH 45433-7765					8. PERFORMING ORGANIZATION REPORT NUMBER AFIT-ENG-DS-17-D-006	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Research Laboratory (Dr. Leah Rowe) 2620 Q. Street WPAFB, OH 45433 leah.rowe.1@us.af.mil 937-938-2552					10. SPONSOR/MONITOR'S ACRONYM(S) 711HPW/RHAS 11. SPONSOR/MONITOR'S REPORT NUMBER(S)	
12. DISTRIBUTION / AVAILABILITY STATEMENT Distribution statement A. Approved for public release; distribution unlimited.						
13. SUPPLEMENTARY NOTES This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States						
14. ABSTRACT Real world environments are non-stationary and include unanticipated changes that contradict what an agent has previously learned. In machine learning, this is called concept drift, a partially-observable change when the environment modifies without notification. Concept drift causes several problems: agents with a decaying exploration fail to adapt, while agents capable of adapting may over fit to noise or overwrite previously learned knowledge. Agents in such environments must take steps to mitigate both problems. This problem is compounded because agents typically have multiple tasks to accomplish simultaneously, requiring multi-objective optimization. This work contributes a novel algorithm that combines concept drift learning with multi-objective reinforcement learning to produce an unsupervised technique for learning in non-stationary multiple objective environments. Testing shows that the agent consistently outperforms traditional multi-objective reinforcement learning. It demonstrates that concept drift can be characterized, detected, and recognized using the developed algorithm.						
15. SUBJECT TERMS multi-objective reinforcement learning, concept drift, non-stationarity						
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT		18. NUMBER OF PAGES	
a. REPORT	b. ABSTRACT	c. THIS PAGE	UU		155	
U	U	U	19a. NAME OF RESPONSIBLE PERSON Dr. Gilbert L. Peterson			
				19b. TELEPHONE NUMBER (include area code) (937) 255-3636, ext 4281; gpeterso@afit.edu		