APPROVAL SHEET

Title of Thesis: Neuroevolution-Based Inverse Reinforcement Learning

Name of Candidate: Karan Kumar Budhraja
MS, Spring 2016

Thesis and Abstract Approved: _____

Tim Oates
Professor
Department of Computer Science and
Electrical Engineering

Date Approved: _____

ABSTRACT

Title of Thesis: Neuroevolution-Based Inverse Reinforcement Learning

Karan Kumar Budhraja, MS, 2016

Thesis directed by: Tim Oates, Professor
Department of Computer Science and
Electrical Engineering

Motivated by such learning in nature, the problem of Learning from Demonstration is targeted at learning to perform tasks based on observed examples. One of the approaches to Learning from Demonstration is Inverse Reinforcement Learning, in which actions are observed to infer rewards. This work combines a feature based state evaluation approach to Inverse Reinforcement Learning with neuroevolution, a paradigm for modifying neural networks based on their performance on a given task. Neural networks are used to learn from a demonstrated expert policy and are evolved to generate a policy similar to the demonstration. The algorithm is discussed and evaluated against competitive feature-based Inverse Reinforcement Learning approaches. At the cost of execution time, neural networks allow for non-linear combinations of features in state evaluations. These valuations may correspond to state value or state reward. This results in better correspondence to observed examples as opposed to using linear combinations. This work also extends existing work on Bayesian Non-Parametric Feature construction for Inverse Reinforcement Learning by using non-linear combinations of intermediate data to improve performance. The algorithm is observed to be specifically suitable for a linearly solvable non-deterministic Markov Decision Processes in which multiple rewards are sparsely scattered in state space. Performance of the algorithm is shown to be limited by parameters used, implying adjustable capability. A conclusive performance hierarchy between evaluated algorithms is constructed.

Neuroevolution Based Inverse Reinforcement Learning

by

Karan Kumar Budhraja

Thesis submitted to the Faculty of the Graduate School
of the University of Maryland in partial fulfillment
of the requirements for the degree of
MS
2016

This work is dedicated to science fiction, and the people determined to convert it into reality

ACKNOWLEDGMENTS

"Dr. Oates is especially interested in understanding the development of the human brainan interest which sparked watching his three daughters grow up".

This translated to me as a multi agent grid world problem. Dr. Oates' interest in the human brain is one of the prominent aspects of his research profile, and also an idea that is motivating. It aligns with my motivation to mimic human thinking and therefore strengthens it.

The idea behind this work is the result of over a year of changing ideas, starting from applications of reinforcement learning. I'm thankful to Dr. Oates for helping me mould the idea all the way through and working with me to identify an niche where this work may be useful.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

The concept of Reinforcement Learning (RL) is motivated by modeling learning by experience. The environment is segmented into states, each of which contain information to describe the environment in that segment. Such information about a particular state comprises a set of attributes or features which can be used to describe or compare states. The learner, also termed as the agent, may benefit differently depending on which state it is in. This creates a notion of rewards corresponding to each state. Reward are conventionally enumerated proportional to the benefit received by the agent on being in a particular state. Based on these rewards, the agent may then develop a plan of actions to take, varied by the state that it is in. Actions may result in the agent moving to a different state, and so a changed environment. A set of actions for each considered state is termed as a policy. For each state, there may be a set of actions available to the agent. In case of a deterministic policy, the agent specifies a single action per state. However, in case of a non-deterministic or stochastic policy, the agent may specify a probability distribution over multiple actions per state. RL then targets to find the optimal policy based on observed rewards. The problem is modeled as a Markov Decision Process (MDP), where the outcome of an action is partly random. Further, the action to be taken at state at a particular time is dependent only on the state at that time. RL is summarized in Figure 1.1. After several iterations of

the algorithm, $\pi \rightarrow \pi^*$.

Consider a reinforcement learning problem where an infant is learning how to assemble blocks into a building. Inferring from reinforcement learning, the infant would engage in many attempts on its own and perhaps eventually learn how to assemble the blocks correctly. In an alternative situation, the infant could also be directed by an adult, where the adult would show the infant how to assemble the blocks and the infant would gather information by watching. At the cost of additional effort, the infant now learns to assemble the blocks potentially faster. The latter situation described is an example of learning from an expert, which is the application area of this work. It is also the way in which humans learn to perform many of the tasks, thereby motivating this class of problems as human-inspired. This work investigates a specific approach to such learning, involving the assumption that state rewards are derivable from state features.

A formal definition of an MDP (Puterman 2014) is repeated here for reference. An MDP is a set of states (S), actions (A) and transition probabilities (θ) between states when an action is taken in a state. Additionally, each state-action pair corresponds to a reward (R). A discount factor (γ) is used while aggregating rewards corresponding to a trajectory of state-action pairs. A policy (π) describes a set of actions to be taken over the state space. The optimal policy (π^*), then, maximizes the expected discounted sum of rewards between two given states ($start$ and $goal$). This stands for the case of an $episodic$ task (which repetitively solves the same problem until a suitable solution is derived). Alternatively, in the case of a $continual$ task (which does not have terminal states), the optimal policy maximizes this sum of rewards over the lifetime of the learning agent. State value (v) is the expected return (sum of R values) when an arbitrary π is followed, starting at that state. The concept of an MDP is extended to define a Linear MDP (LMDP). An LMDP refers to a linearly solvable MDP (using KL-divergence (Kullback & Leibler 1951) or maximum entropy control) (Ziebart 2010). An LMDP is defined by state costs (q) in correspondence

to S as an alternative to R (Vroman 2014). Passive dynamics (p) describes transition probabilities in the absence of control. Following a policy (π) as opposed to p occurs at a cost of the KL divergence between π and p. Such a cost makes the optimization problem convex, removing questions of local optima (Todorov 2006). Additionally, an exponential transformation of the optimal v function transforms the associated Bellaman Equation (Sutton & Barto 1998) (MDP solution) to a linear function. An optimal v function corresponds to a v function evaluated over π^*.

Inverse Reinforcement Learning (IRL) is motivated by learning from examples. As opposed to RL, an agent in IRL does not observe rewards; it attempts to recover them based on an observed policy. The reward is evaluated through observed examples. It is therefore intuitive that IRL is a means to implement Learning from Demonstration (LfD) (Abbeel 2012; Ng, Russell, & others 2000). LfD describes a problem in which an agent learns to perform a task by observing how it is to be done. The observations are in the form of examples, specifically traces of state-action pairs. IRL allows an agent to understand its environment in terms of evaluation of a state. Since state features may individually not be sufficiently informative, it is often required to construct features as their combinations. IRL is summarized in Figure 1.2. After several iterations of the algorithm, $r \rightarrow R$.

One such technique uses regression trees and quadratic programming for this purpose and is described in (Levine, Popovic, & Koltun 2010). Regression trees are tree-like models which distribute a set of observations based on their features. Each node in the tree corresponds to splitting the observations corresponding to that node into smaller groups of observations. Fitting for regression is computed based on correspondence between the demonstration and the policy generated from estimated state rewards. A leaf in the tree corresponds to a numerical value assigned to the group of observations associated with it. On the other hand, quadratic programming is a mathematical formulation of an optimization problem involving a quadratic function of variables. Optimization then involves selection

of a sub-tree without significant loss in regression fitness. Another such technique is based on Gaussian Process (GP) regression (Levine, Popovic, & Koltun 2011b). This technique fits the reward function into a GP kernel as a non-linear combination of state features. Fitting the GP kernel also follows the principle of matching demonstration with the policy generated from estimated state rewards.

Recent techniques have also incorporated a non-parametric Bayesian framework to improve learning. This means that the number of parameters used by the models increase based on the training data. Work in (Ramachandran & Amir 2007) describes the use of priors to create a probability distribution over a set of candidate reward functions. Composite features in (Choi & Kim 2013a) are defined as logical conjunctions of state features. The IRL model extends (Ramachandran & Amir 2007) by defining a prior on these composite features. In (Michini & How 2012; Michini *et al.* 2015), the reward function is additionally assumed to be generated by a composition of sub-tasks to be performed in the MDP space. This algorithm targets detection of sub-goals but does not estimate the final policy over all states in state space (it only targets states observed in the demonstration). This algorithm is therefore excluded from comparison.

In an expectation-maximization based approach, the reward function is modeled as a weighted sum of functions in (Hahn & Zoubir 2015). Parameters of the optimal policy in the model are defined as the probability distribution of actions for each state in the state space, based on the optimal policy. The algorithm then attempts to simultaneously estimate the weights and parameters. The algorithm is not compared with (Levine, Popovic, & Koltun 2011b), but the two algorithms have been individually compared with standard Maximum Entropy IRL. Visual observation of performances of these two algorithms indicates that a GP kernel based approach is competitive to, if not better than, an expectation-maximization based approach. This algorithm is therefore also excluded from comparison.

Very recently, the use of Deep Learning (Deng & Yu 2014) for IRL problems has been explored in (Wulfmeier, Ondruska, & Posner 2015). To summarize, deep learning involves the use of a large number of hidden layers in a neural network structure to obtain significantly abstract concepts (about the input data) as output. The inputs to the first layer of the deep neural network are state features. The performance of this algorithm has been shown to surpass that of existing algorithms ((Levine, Popovic, & Koltun 2011b; Choi & Kim 2013a)). The algorithm focuses on achieving correct expected state value, whereas our algorithm focuses on learning the optimal policy. However, the work is not yet officially published and is therefore unavailable for comparison. Intuitively, it is expected that our work will perform competitively with the use of deep neural networks. The reason for this is that the premise of both algorithms is similar: both algorithms use state features as input to a neural network and evaluate state reward or state value as the output of the neural network. In addition, the use of neuroevolution (evolving the structure a neural network based on task requirements) allows for a more compact network due to dynamic construction of the neural network.

Finally, work in (Vroman 2014) on Maximum Likelihood IRL (MLIRL) is targeted to collectively cover three problem spaces: linear IRL, non-linear IRL and multiple intentions IRL. Linearity and non-linearity is in context to the reward function modeled as a function of state features. Multiple intentions refers to an IRL setting where an MDP comprises of multiple reward functions. The algorithm emphasizes that other IRL methods are not suitable for a unified approach over all the mentioned problem spaces. However, it is mentioned that specialized IRL algorithms are more suitable if number of experts and reward function shape (linear or non-linear) is known. Its performance against other IRL algorithms tested is concluded as competitive. Performance of MLIRL for our problem setting is therefore evaluated as at most competitive with (Levine, Popovic, & Koltun 2011b) (evaluated in (Vroman 2014)). MLIRL is conclusively excluded from comparison for our

work.

As opposed to evaluation of states to rewards as in (Levine, Popovic, & Koltun 2010), this work proposes and evaluates an evolutionary neural network based approach to LfD based on generation of state values from state features and their combinations. Evolution of neural networks networks for LfD is analogous to evolution of neurons and connections in the human brain.

The use of neural networks allows for inherent advantages over regression trees (Caruana & Niculescu-Mizil 2006; Bengio 2009). Unlike regression trees, neural networks are capable of learning non-linear data boundaries. They are able to generate more abstract features at hidden neurons. Neural networks are also less prone to overfitting than regression trees. Finally, the fact that neural networks can approximate any function with sufficient data (universal approximators) makes them intuitively preferable.

Further, when fitting a highly non-uniform function, neural networks are better than kernel functions (used in GP regression) at generalizing non locally and scaling to larger datasets. This is because kernel functions typically generalize locally.

Chapter 2 describes relevant work in the context of neuroevolution and feature based IRL. Chapter 3 discusses IRL and a feature based approach to the problem. This is followed by details of the proposed neuroevolution based algorithm in Chapter 4. Chapter 5 on experimental evaluation of the algorithm against FIRL is proceeded by the concluding remarks in Chapter 6.

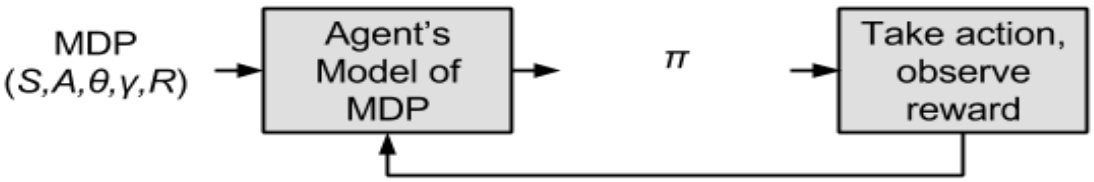

FIG. 1.1. Reinforcement Learning

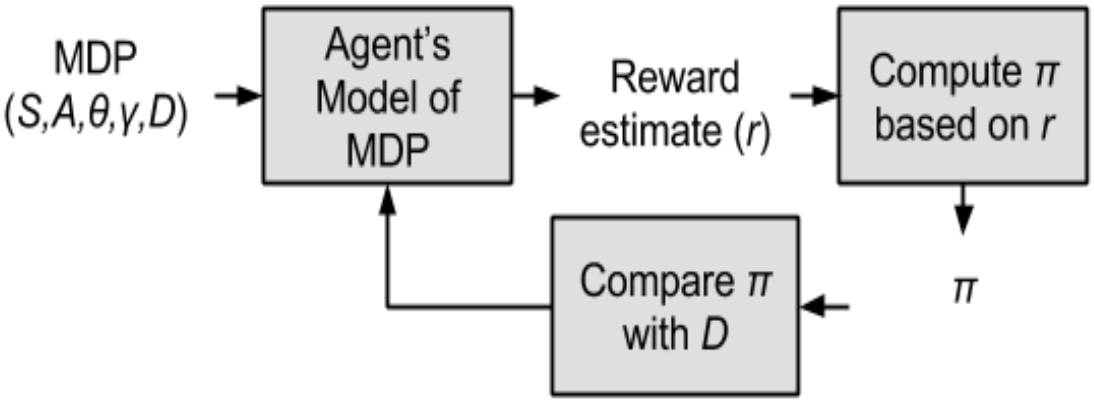

FIG. 1.2. Inverse Reinforcement Learning

Chapter 2

RELATED WORK

Feature construction for IRL (FIRL) (Levine, Popovic, & Koltun 2010) uses logical combinations of state features to generate more informative features. These features are then mapped to a reward function as a weighted sum. This reward function is used to generate a policy for the given state space. (Levine, Popovic, & Koltun 2010) examines the superior performance of FIRL against IRL models of Linear Programming Apprenticeship Learning (LPAL) (Syed, Bowling, & Schapire 2008), Maximum Margin Planning (MMP) (Ratliff, Bagnell, & Zinkevich 2006), and Abbeel&Ng (A&N) (Abbeel & Ng 2004) and their variants.

In (Levine, Popovic, & Koltun 2011b), FIRL is acknowledged to have limited capabilites in representing a reward function which uses a non linear combination of features. This limitation is overcome by learning a reward function using GP regression. Experimental results in (Levine, Popovic, & Koltun 2011b) demonstrate superiority of Gaussian Process IRL (GPIRL) over prior IRL methods such as those evaluated in (Levine, Popovic, & Koltun 2010). Additionally, GPIRL is also evaluated against Maximum Entropy IRL (MaxEntIRL) (Ziebart *et al.* 2008), Multiplicative Weights for Apprenticeship Learning (MWAL) (Syed & Schapire 2007), Maximum Margin Planning Boost (MMPBoost) (Ratliff *et al.* 2007) and LEArning to seaRCH (LEARCH) (Ratliff, Silver, & Bagnell 2009) and

their variants.

Bayesian Non-Parametric Feature construction for IRL (BNP-FIRL) (Choi & Kim 2013a) uses the Indian Buffet Process (IBP) (Ghahramani & Griffiths 2005) to define priors over composite state features. The IBP defines a distribution over infinitely large (in number of columns) binary matrices. It is used to determine the number of composite features and the composite features themselves. Features and corresponding weights are recorded over several iterations of the algorithm. These values are then either aggregated (as a mean-based result) or used for estimating Maximum A-Poteriori (MAP) (Choi & Kim 2011) (as a MAP-based result) to evaluate state reward values. In brief, MAP is technique which uses empirical observations to estimate the value of an unobserved quantity. While the two results provide competitive performance when calculating reward function, using the mean-based result performs significantly better than the MAP-based result when focusing on action-matching rather than reward-matching. For this reason, MAP-based results are excluded from comparison. More importantly, this emphasizes the importance of how the iteration data is finally used. A non-linear combination of this data intuitively provides better performance than a linear combination (as in the case of mean). Experimental results in (Choi & Kim 2013a) indicate superiority of BNP-FIRL over GPIRL in a non-deterministic (a certain percentage of actions are random) MDP, such as the one used to evaluate our work.

On the lines of FIRL, it is proposed that a neural network provide a mapping from state features to state value. Since a neural network can compactly contain complex combinations of inputs, internal neurons may represent more informative features, as those obtained in FIRL. In an alternative implementation extending BNP-FIRL, a neural network is used to provide state value based on feature and weight data gathered over several iterations of state reward estimation in the BNP-FIRL algorithm.

Since the function to be generated is unknown, so is its complexity. This implies

uncertainty about the optimal number of layers and nodes in each layer to be used. Nodes in later layers of a neural network may be able to define a function for which it may take several nodes in the earlier layers of the neural network to define. There is therefore a trade-off between the number of hidden layers and number of nodes in each layer. Because of this, using a fixed structure for the neural network in this scenario may be difficult and sub-optimal. Neuroevolution solves this problem by generating the optimal neural network using techniques such as Genetic Programming (GP) (Gruau & others 1994), Evolutionary Programming (EP) (Angeline, Saunders, & Pollack 1994), Simulated Annealing (SA) (Yao & Liu 1997), Genetic Algorithms (GA) (Stanley, Bryant, & Miikkulainen 2005b; Stanley & Miikkulainen 2002), Evolution Strategies (ES) (Kassahun & Sommer 2005; Siebel & Sommer 2007), Evolutionary Algorithms (EA) (Rempis 2012) and Memetic Algorithms (MA) (Sher 2012).

In a $direct\ encoding$ scheme such as that employed by NEAT, all neurons and connections in the neural network are explicitly specified by the genotype. In case of an $indirect\ encoding$ scheme, these values are expressed implicitly by smaller parts of the genotype (encoding of the neural network). $Indirect\ encoding$ is suitable for solving tasks which have a high degree of regularity (such as controlling the legs of a millipede like robot). Approximating a function is a problem that lacks regularity. A well established $direct\ encoding$-based neuroevolution technique such as NeuroEvolution of Augmenting Topolgies (NEAT) (Stanley & Miikkulainen 2002; Stanley 2014), which evolves both the structure and parameters of the neural network, is therefore preferred for our work.

NEAT evolves a population of neural networks governed by GA (Koza 1992; Banzhaf *et al.* 1998), and therefore a corresponding fitness function. There currently exist many extensions of NEAT, including rtNEAT (Stanley, Bryant, & Miikkulainen 2005b) (a real-time version of NEAT, which enforces perterbation to avoid stagnation of fitness level) and FS-NEAT (Whiteson *et al.* 2005) (NEAT tailored to feature selection).

In (Yong *et al.* 2006), demonstration bias is introduced to NEAT-based agent learning. This is done by providing advice in the form of a rule-based grammar. This does not, however, provide an embedded undertanding of preference for a state in state space. This is further explored by mapping states to actions in (Karpov, Valsalam, & Miikkulainen 2011) where different methods of demonstration are studied in a video game environment (Stanley, Bryant, & Miikkulainen 2005a). NEAT-based IRL is also implemented in a multiple agent setting as in (Miikkulainen *et al.* 2012) where groups of genomes share fitness information.

State values are used to generate a corresponding policy. To incorporate learning by demonstration, this policy is matched with the demonstration and the neural network is evolved thereof. Such directed neuroevolution can be considered as an extension of (Yong *et al.* 2006; Karpov, Valsalam, & Miikkulainen 2011) with better insight to evaluation of a state. For the purpose of this document, the proposed work is referred to as NEAT-IRL.

Chapter 3

PROBLEM DEFINITION

The goal of IRL is to learn a reward function for each state in a given state space based on parts of a given policy (demonstration). In a broader sense, the goal is to be able to generate a policy over a state space (S), which is correlated to what has been demonstrated. For the purpose of this work, the demonstrations received by algorithm are assumed to be performed by an expert, meaning that they are assumed to be optimal. A demonstration (D) consists of numerous examples, each of which is a trace of the optimal policy through state space. These are represented in the form of state-action pairs (s, a).

One method to generate a policy is to generate state values for each state based on state features. It is assumed that a weighted combination of state features and their combinations can provide a quantitative evaluation of a state (with a motivation similar to FIRL). The first problem, then, is to learn a mapping from state features to state values, which produces a policy for which state-action pairs are consistent with the given examples.

Additionally, values of weights and features over several iterations of BNP-FIRL may be used in different ways to derive a state reward value. The second problem, then, is to learn a non-linear mapping from these values to state reward, which produces a policy consistent with the given examples (as described for the first problem).

The state space domain used for this work is a grid world Markov Decision Process

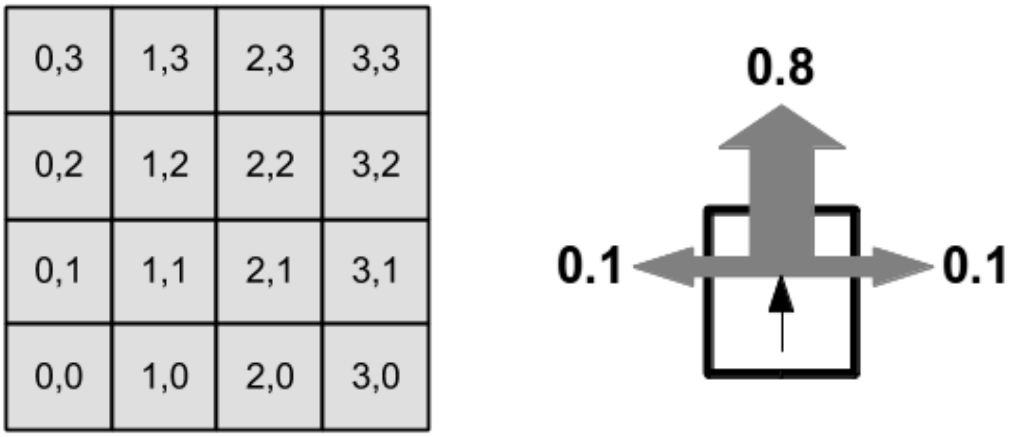

FIG. 3.1. An example of MDP state features and transition model

(MDP), but the concept can be extended to other state spaces. The use of a grid world MDP is aligned with experiments in (Levine, Popovic, & Koltun 2010; Michini & How 2012; Michini *et al.* 2015; Vroman 2014; Hahn & Zoubir 2015). In case of a grid world MDP, an agent has 5 possible actions: move up, move down, move left, move right, do nothing. In case of a deterministic MDP, the action taken is always the action selected. However, in a non-deterministic MDP, the action taken is sometimes random, irrespective of the action selected. Apart from such randomness, the agent may also move in a certain direction other than that specified based on probability. An example of state features (cell coordinates in the example) and state transition model is given in Figure 3.1. The bottom left-most cell state is assumed to be at the origin. In the canonical example of transition probabilities shown, the agent selects to move up. On doing so, the agent has an 80% probability of moving up and a 10% probability each of moving in directions perpendicular to the intended direction. Note that in this example, the agent has a 0% chance of taking an absolutely random action at a given state.

Chapter 4

PROPOSED METHOD

As in Figure 4.1, examples are provided as traces of subsequent states in the state spaceas per demonstration policy. Multiple traces may overlap, which means that a single state may be covered more than once in a set of examples. If no examples overlap, the set of states involved in the examples therefore has an upper bound of the total number of examples multiplied by the length of each example (in states). An example from an MDP policy perspective is given in Figure 4.2. These examples serve as demonstration (D) in IRL (used for learning state rewards).

The following sections provide an overview of fundamental algorithms in the context of this work and then continue to define an approach to NEAT based IRL.

FIG. 4.1. Demonstration generation

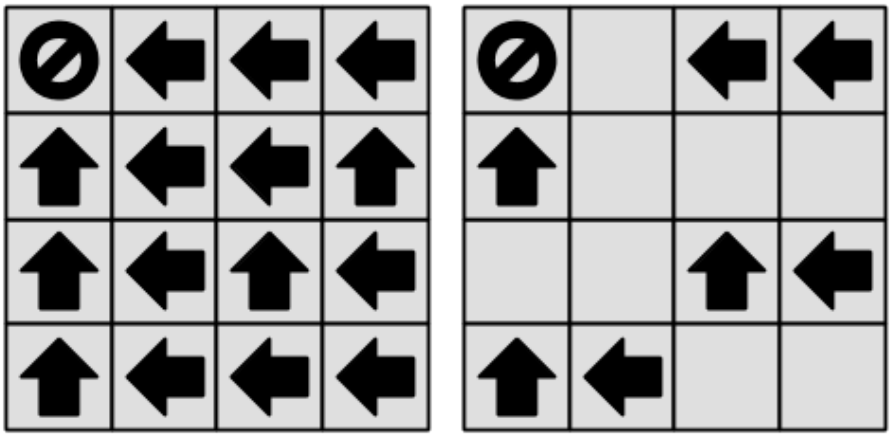

FIG. 4.2. Policy sampling: 4 demonstrations of length 2

4.1 FIRL

The methodology of FIRL is summarized in Figure 4.3. There are two steps which are repeated in each iteration: $optimization$ and $fitting$ (Levine, Popovic, & Koltun 2010). In the $optimization$ step, a reward function R is generated that best fits the current feature set ϕ, while evaluating to a policy consistent with D. This is done by minimizing the squared error between R and R', R' being the linear projection of R on ϕ. In the $fitting$ step, a regression tree of R is built over S. The smallest tree which is consistent with D is used. Leaf clusters of this tree are then used to represent new features.

4.2 GPIRL

GP based regression (Williams & Rasmussen 2006) is used for value function approximation for RL in (Deisenroth, Rasmussen, & Peters 2009; Engel, Mannor, & Meir 2005; Rasmussen, Kuss, & others 2003). GPIRL (Levine, Popovic, & Koltun 2011b) tailors this model to learn reward structure (based on state features) as GP kernel hyper-parameters (θ). Unlike GP regression, GPIRL does not assume that the demonstration may be sub-optimal.

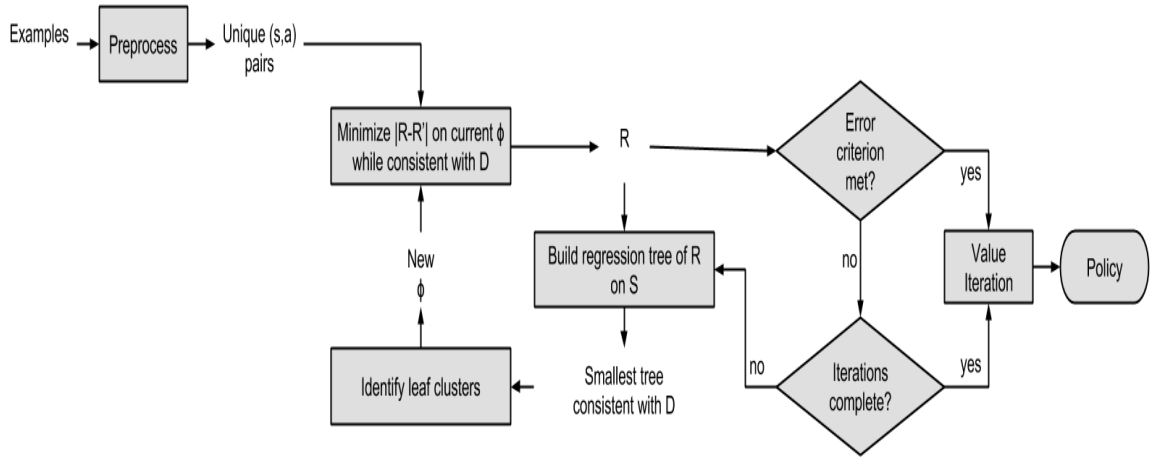

FIG. 4.3. FIRL summary

Demonstration observations (u) and θ are calculated by maximizing their likelihood, given the demonstration. The kernel is regularized to prevent trivial solutions in the form of singular covariance matrices. If the number of states in the demonstration is large, then only a subset is used for GPIRL to maintain tractability. As opposed to FIRL, GPIRL generates a policy containing a probability distribution over possible actions at a state. This can be made deterministic by greedy action selection based on their probabilities.

4.3 BNP-FIRL

BNP-FIRL is summarized in 4.4. BNP-FIRL decomposes reward (r) as a product of composite features (Φ) and weights (w). There are a total of K composite features and so the size of w is K. Each column in Φ is a binary vector of size $|S|$, indicating presence or absence of the feature per state. The size of Φ is therefore $K \times |S|$. The product of w and Φ is thus a vector of size $|S|$.

Ψ denotes a matrix of M atomic features (the original state features as defined by the MDP). As for Φ, each column in Ψ contains $|S|$ binary data items, making its size $M \times |S|$.

The matrix Z indicates the atomic features that comprise each composite feature. It

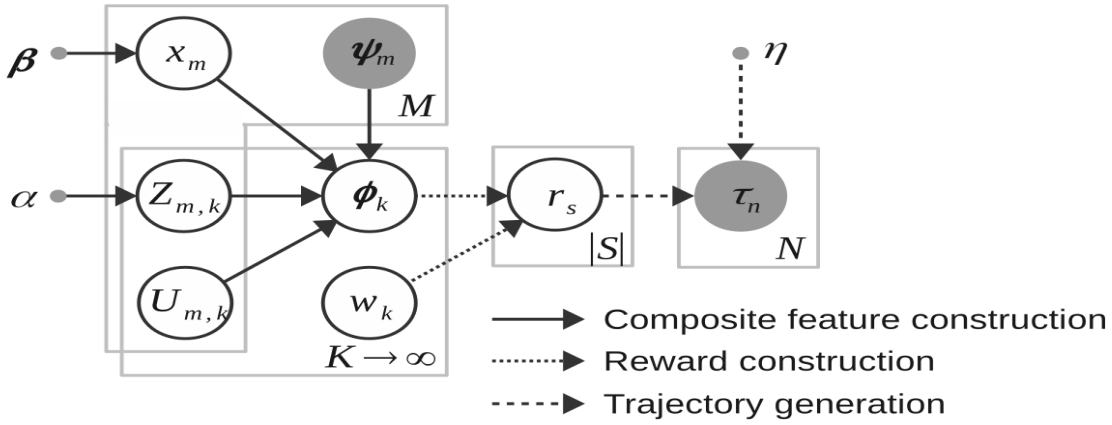

FIG. 4.4. BNP-FIRL summary
(Choi & Kim 2013a)

is therefore $M \times K$ in size. The matrix U (also $M \times K$) is a binary matrix, the Bernoulli distribution (with $p = 0.5$) which causes atomic features to be negated. Negation is in context of the use of the binary value in the formula used to generate the composite feature. X is an M-dimensional binary vector indicating use of atomic features to form composite features.

α is a constant parameter over which a Poisson distribution is calculated (required by the IBP when computing the number and values of composite features). A Bernoulli distribution ($p = \kappa$) is used to generate priors on X. Finally, κ is Beta distributed using the closed interval defined by β.

τ is vector of N demonstrations that is assumed to be generated using the optimal policy for the MDP. The posterior probability of τ given r and constant η is then computed. The algorithm iteratively converges to a value of r which maximizes this probability. The values of Φ and w computed in each iteration are stored in a vector. To compute mean-based results, the sum of rewards per iteration is used as the final reward. The use of mean at the final stage of the BNP-FIRL algorithm is denoted by BNP-FIRL(mean).

4.4 Genetic Algorithms

GA (Koza 1992; Banzhaf *et al.* 1998) is a biologically inspired population based stochastic search technique. The search is directed towards optimizing a $fitness\ function$. The input variables of this function form the search space. Each $genome$ represents a combination of input variable values. This is visualized as a point in the search space. Each iteration (or $generation$) of GA primarily comprises of two operations: $crossover$ and $mutation$. $Crossover$ refers to an operation where two existing $genomes$ are used to create new $genomes$ by merging parts from the existing $genomes$. $Mutation$ refers to an operation in which parts of an existing $genome$ are randomly modified to create new $genomes$. In this manner, the pool of $genomes$ is increased. To maintain a fixed population, $genomes$ are then evaluated on the $fitness\ function$ and the pool is truncated to keep the best $genomes$. The performance of GA is primarily governed by two parameters: population size (N_P) and maximum number of generations (N_G).

GA is summarized in Figure 4.5, where each cell in a genome is a positive integer. Each genome corresponds to an arbitrary fitness value (denoted in bold). The existing population of genomes is brought down to the original size by selection based on these fitness values. The next two sets of genomes represent the outcomes of $crossover$ and $mutation$.

4.5 Artificial Neural Network

An Artificial Neural Network (ANN) or a Neural Network (NN) (Haykin & Network 2004) is a function approximation model inspired by biological neural networks. The model is composed of interconnected nodes or $neurons$ which exchange information to produce the desired output. As summarized in Figure 4.6, an NN is traditionally segmented into three layers: input, output and hidden. Arrows denote the direction of information

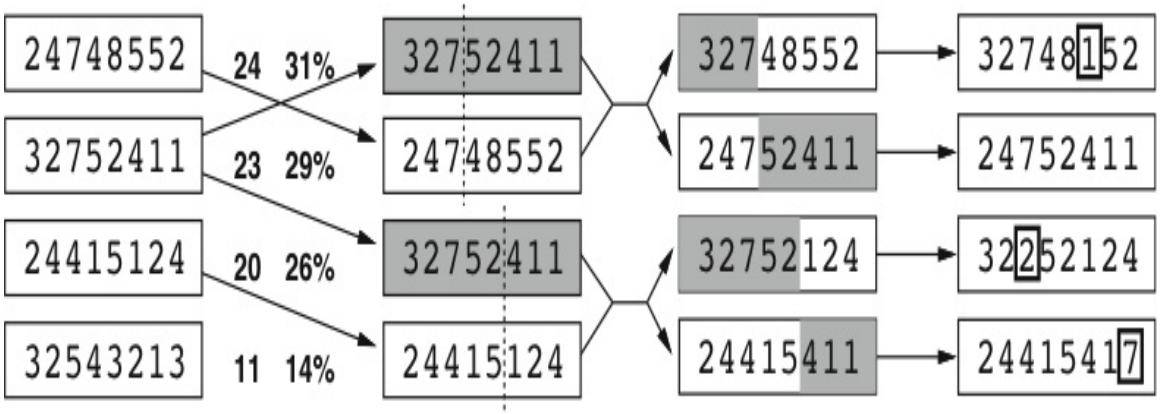

FIG. 4.5. GA summary
(Tufts University 2015)

flow. The input layer corresponds to values (X) of input variables being provided to the NN. The output layer corresponds to values (Y) of output variables being generated by the NN. Nodes in the hidden (from the user) layer therefore contain meta-data used to generate the final output. The NN shown in Figure 4.6 is termed as a $single\ layer$ NN based on the number of hidden layers in the NN.

A node in each layer in the NN is evaluated by combinations of information from nodes in the previous layer. Weights (w) are used to combine information from different nodes in the previous layer. Bias (b) is a constant valued input to a node. An NN may therefore be parametrized based on the number of inputs and outputs, number of hidden layers and the size of each layer. These can be derived from a specification of all the w and b values.

In our work, we use neural networks containing a single output node.

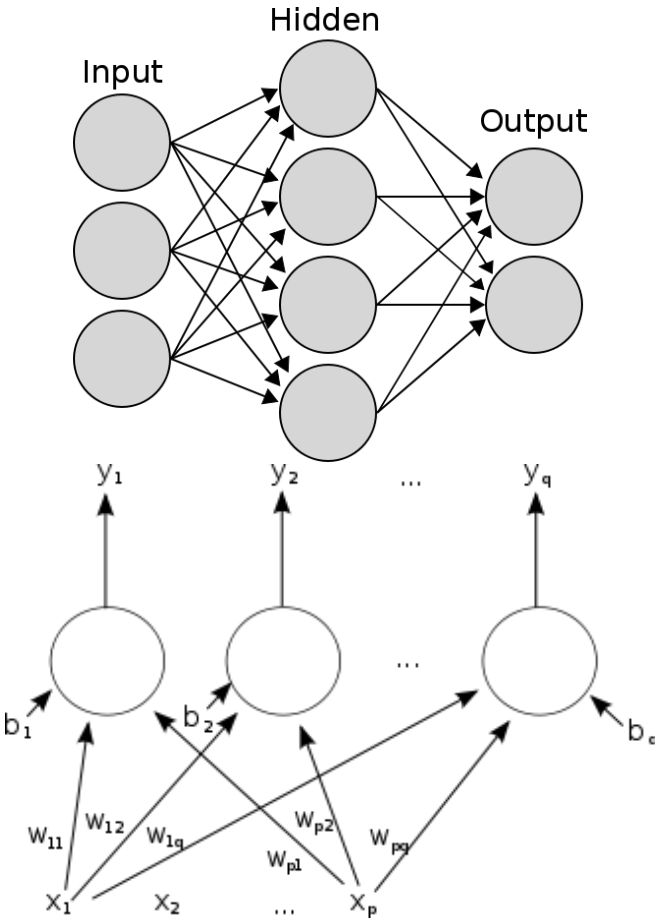

FIG. 4.6. Artificial neural network
(Wikipedia 2015)

4.6 NEAT

NEAT evolves neural networks using GA, guided by a $fitness\ function$. Each member of the population corresponds to a genotype or genome and a phenotype (the actual neural network). NEAT begins with relatively less complex neural networks and then increases complexity based on $fitness$ requirement. There are primarily two parameters which affect the performance of NEAT: N_P and N_G, both of which are inherent to GA. They specify the number of genomes (maintained at the end of each generation of the algorithm) and the maximum number of generations for which the genomes are to be evolved.

4.7 NEAT-IRL

The result of NEAT-IRL is a neural network which can produce state values based on state features.

Neural networks represented by a genome population in NEAT are considered to use state features as input and produce state value as output. A corresponding policy is evaluated for that state space. It is therefore suitable that the fitness function be the coherence between the generated and demonstrated policies. In implementation, this is done by using the coherence of generated action directions with the demonstration as fitness value. Thus, an action in direction opposite to that in the example is penalized more (fitness value -1) than one which is perpendicular to it (fitness value 0). These values are intuitive to the cosine of the angle between the generated and example actions. Fitness is accumulated over all states included in the demonstration. If all of the demonstrated actions are replicated correctly, the algorithm is terminated. This is summarized in Figure 4.7.

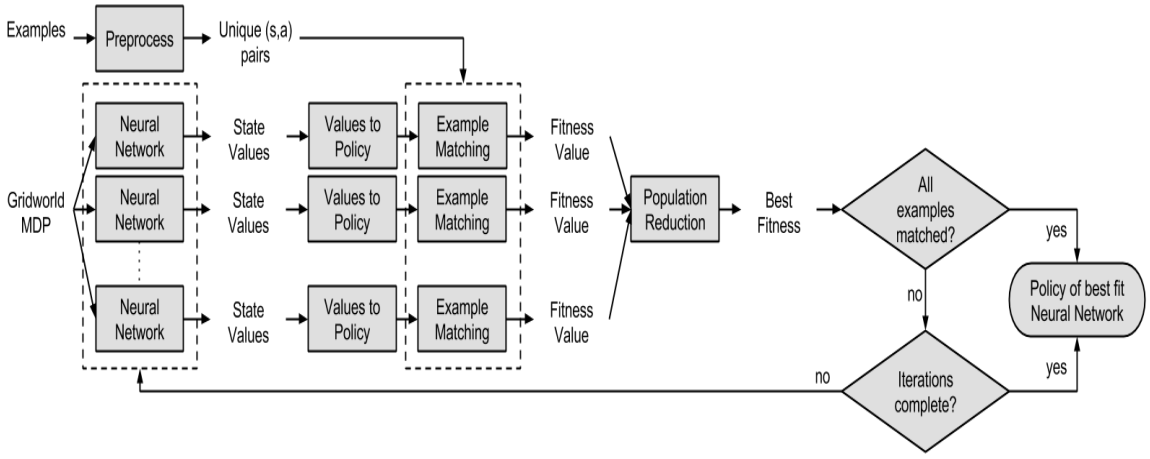

FIG. 4.7. NEAT-IRL summary

4.8 BNP-FIRL(NEAT)

The use of NEAT is similar to NEAT-IRL where the fitness value is based on matching examples in the demonstration. However, the inputs are no longer state features.

The dimension of Φ and thus w vary across iterations, because of the continuous variation in composite features considered. A non-linear combination of Φ or w over time is therefore not possible. However, the size of r is constant for each iteration. The set of values of r over iterations of the algorithm then serve as input to the neural network. The output of the neural network is then used as the resultant state reward vector. This reward vector is used to compute the optimal policy, which can then be used to compute fitness values.

4.9 FIRL and GPIRL vs NEAT-IRL

FIRL, GPIRL and NEAT-IRL all consider individual states as opposed to traces of state sequences, making them memory efficient in terms of the storage of examples. However, FIRL and GPIRL generate a function which produces state rewards, as opposed to

NEAT-IRL which produces state values. State rewards evaluate $immediate$ desirability whereas state values evaluate long term desirability of a state. An agent therefore seeks actions which lead to states of higher value and not highest reward to maximizes long term reward (Barto 1998). State values also observe smoother transition amongst states in close proximity, though this is not necessary for state rewards. Further, converting a set of state values to a policy requires a single computational step (value based greedy action selection). In case of state rewards, however, the policy is accumulated over several $epochs$ of computation, thereby resulting in greater time complexity. Additionally, a policy generated using state values is more robust to noise (in context of real-world applications such as robotics) than one generated based on state rewards. The reason is that in case of state values, policy evaluation only requires a comparison of adjacent state values. The final policy is therefore only affected if the state values are close enough for noise to change the action selection. In case of policy generation using state rewards, noise would be accumulated over each $epoch$ of computation and would therefore have a more significant effect on the generated policy. A disadvantage of generating state values is that they transfer poorly to other MDP with similar feature sets as opposed to state rewards (Vroman 2014). However, our problem space does not concern transfer learning.

Additionally, FIRL involves convex optimization (minimization) and the use of regression trees, whereas NEAT-IRL is built on neural networks.

(Lilley & Frean 2005) shows that GP behaviour can be replicated by a multi layer perceptron neural network with a sufficiently (tending to infinity) large number of hidden neurons. This stands with a requirement to use weight decay (Neal 1996; 1995). In practice, this implies limitation of neural networks for approximation of GP behaviour instead of exactness. Further, GP models can be optimized to fit data exactly with specific hyperparameter values (Rasmussen 2006). This implies a trade-off between exactness and overfitting data.

NEAT-IRL does, however, introduce two new parameters (N_P and N_G) to the IRL problem, which increases degrees of freedom. The performance of a fixed set of parameters will vary in different environments. Algorithm performance may therefore need to be evaluated across these parameters for optimal value assignments.

4.10 BNP-FIRL(mean) vs BNP-FIRL(NEAT)

The set of functions defined by a linear combination of variables is a subset of a the set of functions defined by a non-linear combination of those variables. Non-linear combinations are therefore more powerful in expressing relationships among variables and direct towards better function approximations at the cost of function complexity. In the case of the mean-based result of BNP-FIRL in particular, the linear combination is simply a sum of variables. This allows for significant scope for improvement in approximation of the final value of state rewards.

Similar to NEAT-IRL, the use of NEAT with BNP-FIRL increases the number of algorithm parameters and may require an additional level of optimization for better algorithm performance.

Chapter 5

EXPERIMENTS

Originally conceptualized in a Python implementation based on MultiNEAT (Chervenski 2012), NEAT-IRL is currently implemented in MATLAB using (Mayr 2003) and is evaluated using existing tools in the IRL toolkit containing FIRL and GPIRL (Levine, Popovic, & Koltun 2011a). The implementation of BNP-FIRL exists in an extended version of the toolkit (Choi & Kim 2013b). It is copied to and used with (Levine, Popovic, & Koltun 2011a) for the experiments in this work. Inherent to the toolkit, there exist $2\times(n-1)$ binary state features for a grid of size n. These bit patterns contain sub patterns which are consistent for a row or column in the grid, thereby forming a coordinate system. This is exemplified in Figure 5.1. Additionally, state rewards are assigned randomly for each macro block.

FIRL and GPIRL consistently produce better results than the other IRL algorithms they are compared to in (Levine, Popovic, & Koltun 2010; 2011b). The case is similar with BNP-FIRL(mean) (Choi & Kim 2013a). Additionally, GPIRL performs consistently better than FIRL. It is therefore sufficient to evaluate GPIRL, NEAT-IRL, BNP-FIRL(mean) and BNP-FIRL(NEAT) when examining performance improvements. NEAT-IRL is not compared with the work done in (Yong *et al.* 2006; Karpov, Valsalam, & Miikkulainen 2011) in sight of a dependency on rule based learning.

The algorithms are evaluated in a number of ways. NEAT-IRL is evaluated individually and also in comparison to GPIRL, BNP-FIRL(mean) and BNP-FIRL(NEAT). The IRL toolkit (Levine, Popovic, & Koltun 2011a) defines misprediction score as the probability that the agent will take a non-optimal action (different from what would have been in an example) in that state space. This is measured based on matching of expert policy (used for demonstration) and the policy generated by the IRL algorithm. These scores are evaluated for both linear and standard MDPs. With 4 possible actions at any state in a grid world, default misprediction score is 0.75. Marcoblock size, b, which specifies a number of adjacent cells in a grid to be assigned the same reward value, is set to 1 for all experiments. This is done so that state features correspond to unique rewards. Average values are computed over 25 executions. Furthermore, NEAT-IRL may end execution early when a generated policy completely matched with demonstrated examples. This may, however, lead to under-fitting, which is also a possible contributor to hindering the performance of NEAT-IRL in terms of misprediction score.

Note that the default values used for NEAT in (Mayr 2003) are $N_P = 150$ and $N_G = 200$. Additionally, GPIRL is evaluated in (Levine, Popovic, & Koltun 2011a) using a default grid size (n) of 32, i.e. a 32×32 grid with 16 training samples (N_S) of length (L_S) 8 each. Reduced values (scaled in proportion) are used for evaluation of NEAT-IRL and BNP-FIRL(NEAT) for computational tractability. Primary results discussed in this work use the configuration $n = 16$, $N_S = 8$ and $L_S = 4$. Certain supplementary evaluations use the configuration $n = 4$, $N_S = 4$ and $L_S = 1$. The number of samples in this case is more than the scaled value ($N_S = 2$) to avoid inference based on little data. In doing so, the demonstrations include 25% of the total states, which is justified to experiment on considering the proportions used in the original setting ($n = 32$). The primary grid allows for standardized evaluation of the algorithms, whereas a smaller grid allows more tractable analysis of the MDPs over which the algorithms are evaluated. GPIRL parameters are used

as default in (Levine, Popovic, & Koltun 2011a). Note that smaller data samples, interpolation used by GPIRL exceeded the possible number of points, thereby causing mathematical errors. This was fixed by limiting interpolation to the maximum possible, in case the number of interpolations was more than what was possible.

In the misprediction score graphs plotted, a solid line is used to denote performance data on a standard MDP and a dotted line is used to denote performance data on a linear MDP. Additionally, the term computational complexity is interchangeably used with computational time complexity and execution time is calculated in seconds. Additionally, misprediction score is observed to be lower when testing on a linear MDP than on a standard MDP irrespective of which of these two MDP types was used for training the algorithm.

5.1 NEAT-IRL

Experimental evaluation begins with testing NEAT-IRL parameters N_P and N_G. This is done to evaluate the effect of NEAT parameters on algorithm performance and complexity.

Figure 5.2 shows variation of misprediction score and execution time. The value of N_P is varied from 15 to 150, while N_G is arbitrarily set to 50. As expected in the use of GA, a larger population of genomes provides better performance. Note that beyond a threshold value, an increase in population does not contribute to significant optimization. This is because the capacity of search that can be performed by the extra (beyond threshold) population of genomes is limited by the size of the current state space. Noise in the graph pertains to randomness of the involved processes. Execution time for NEAT-IRL is linear with population size. Since the implementation of the algorithm is currently not parallel, parallelization of executions for each member of the population can theoretically provide constant execution time with respect to population size. This is possible because executions

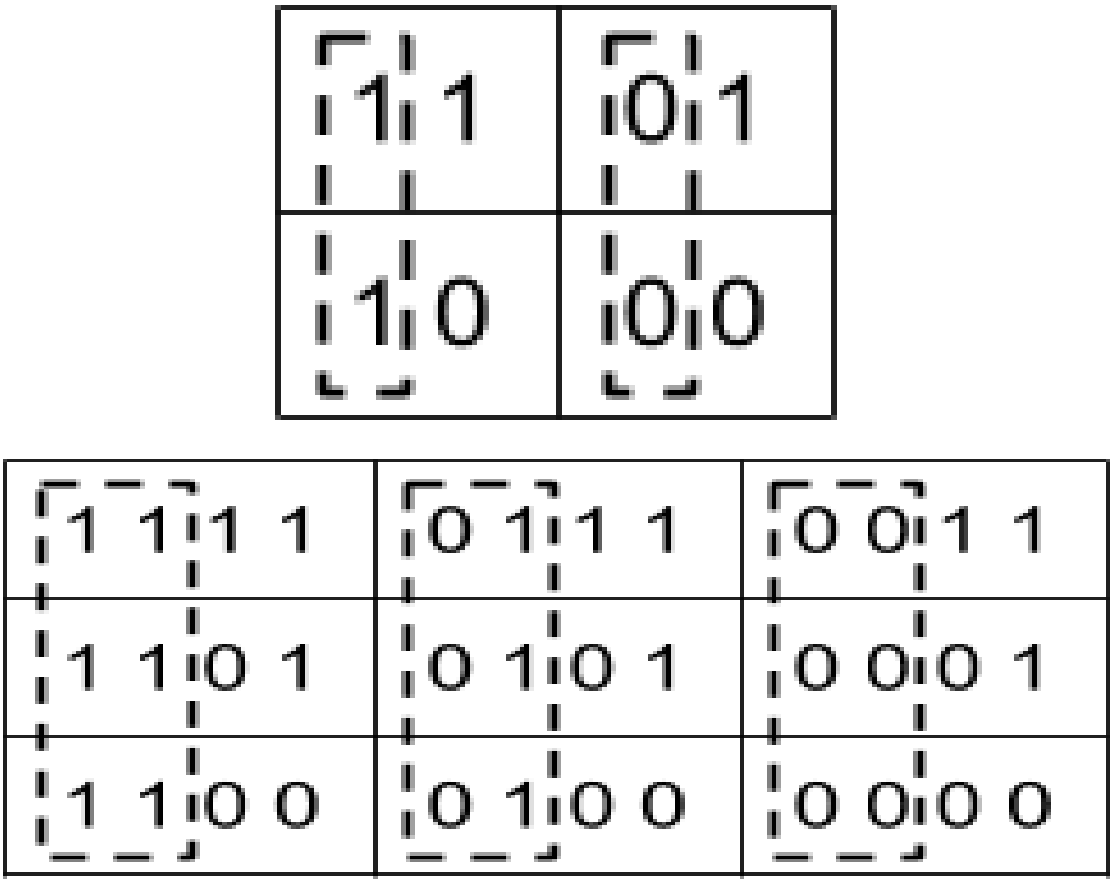

FIG. 5.1. State features ($n = 2$, $n = 3$)

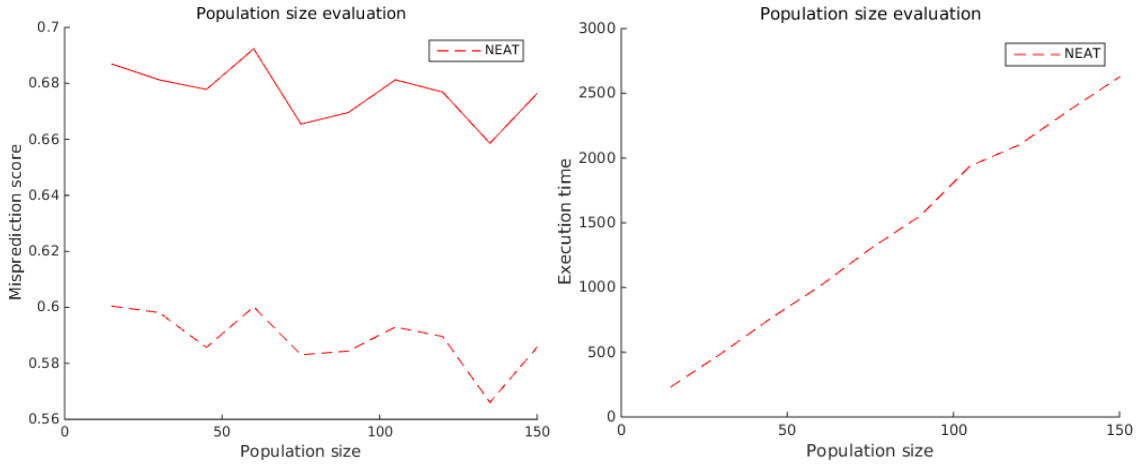

FIG. 5.2. NEAT-IRL population size evaluation (linear MDP, $d = 0.1$)

corresponding to various agents in the same generation are independent of each other.

In a similar manner, Figure 5.3 evaluates NEAT-IRL on the number of generations. The value of N_G is varied from 2 (to allow at least one stage of gene mutation and crossover) to 10 using $N_P = 150$. Coherent to GA behaviour, an increase in the number of maximum generations results in lower misprediction score. This is subject to stagnation in a manner similar to that discussed for 5.2. NEAT-IRL execution time is linear with the maximum number of generations as expected, since computation is constant per generation of the algorithm.

Figure 5.2 and Figure 5.3 correspondent to linear and standard MDPs respectively. The reason is that the results are more pronounced in either situation. The MDPs used are completely deterministic.

5.2 GPIRL and BNP-FIRL(mean) vs NEAT-IRL and BNP-FIRL(NEAT)

GPIRL, BNP-FIRL(mean), NEAT-IRL and BNP-FIRL(NEAT) are compared across three aspects of the MDP: the MDP type (standard or linear), the amount of determinism in the MDP (d, where a value of 1.0 represents complete determinism) and different values

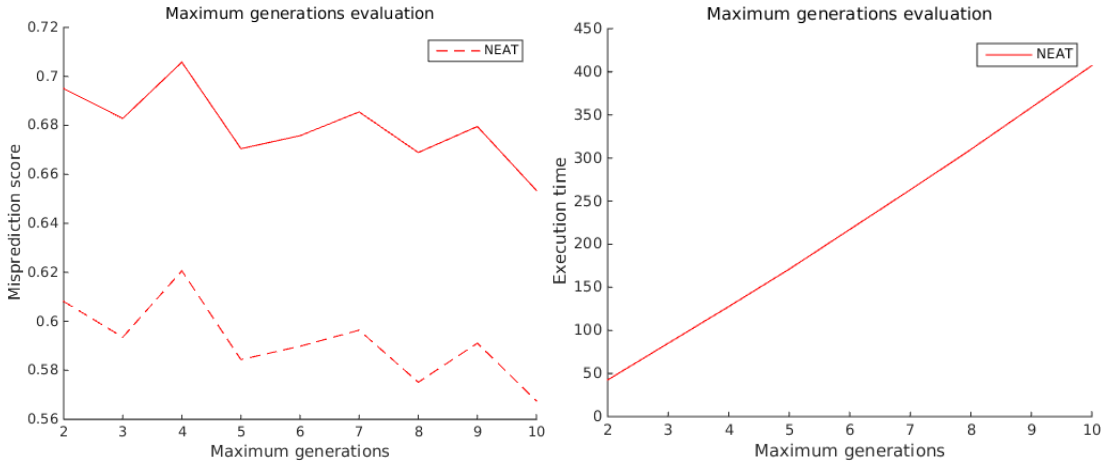

FIG. 5.3. NEAT-IRL maximum generations evaluation (standard MDP, $d = 0.1$)

of N_S. This tests the ability of the algorithms to reconstruct the reward function across different amounts of data. For each of these evaluations, NEAT parameters are arbitrarily set as $N_P = 50$ and $N_G = 50$.

Dependency on N_S is evaluated by varying N_S from 1 to 8. To evaluate dependency on d, performance of the algorithms in two settings ($d = 0.7$ as in (Choi & Kim 2013a) and $d = 1.0$) are tested. Both of these settings are tested on standard and linear MDPs. These experiments are depicted in Figure 5.4, Figure 5.5, Figure 5.6 and Figure 5.7.

In context of misprediction score, the performance of algorithms using neural networks is more competitive with the compared algorithms in a non-deterministic setting than in a deterministic setting. This favours use of the algorithm in a real world setting where non-determinism exists because of various sources noise. It is also the case that better performance of neural networks is observed for a linear MDP than for a standard MDP. This is attributes to the neural network being able to perform better on a more easily solvable MDP given a set of parameters. Additionally, the composite features and thus reward function over iterations provide better performance of neural networks in BNP-FIRL(NEAT) as compared to the use of state features in NEAT-IRL. As in (Choi & Kim

2013a), BNP-FIRL(mean) is observed to be consistently better than GPIRL.

Performance of NEAT-IRL (the purely NEAT based algorithm) improves at a slower pace than the other algorithms. This is attributed to the values of $N_P = 50$ and $N_G = 50$ being unoptimized for the MDPs being used. It is demonstrated in section 5.1 that these values are tunable to improve the performance of the algorithm.

Execution time is least for GPIRL in all four experimental settings. In a trade-off with better performance, an increased execution time is observed for BNP-FIRL(mean). This increase is more significant in the case of $d = 0.7$. This implies that a linear MDP is a harder problem for BNP-FIRL(mean) to solve. The additional layer of NEAT at the final stage of BNP-FIRL does not introduce noticeable increase in execution time. Note that this is also because of the scale being used to include much larger values, which results in lower resolution between the two graphs.

Execution time for NEAT-IRL remains largest across all of the experimental settings. In the presence of less demonstration data, the algorithm may often match all examples and discontinue evolving the network further. As the number of examples increase, fitting becomes more difficult and causes an increase in the increased number of generations. This explains the gradual increase in time complexity. Since we have a limit on the maximum number of generations that may be executed, the time complexity later stagnates. BNP-FIRL internally uses a matrix multiplication based technique to solve the MDP based on state calculated rewards. However, NEAT-IRL processes each state based on state values of neighbouring state values. It is therefore possible that the additional conditional sequences result in significantly increased time complexity for NEAT-IRL as opposed to BNP-FIRL(NEAT). Perhaps, then, if we use neural networks to generate state reward instead of state value, we could integrate with the MDP solution method used by BNP-FIRL and reduce time complexity. However, mapping from state features to rewards is argued to decrease performance as compared to mapping from state features to state values (as

discussed in Section 4.9).

From a performance improvement perspective, Figure 5.6 examines a setting where the use of neural networks provides competitive performance to the other algorithms in our examination set. However, a smoother graph is required for better comparison. For this reason, an average is considered over 100 executions. The results are shown in Figure 5.8. The figure establishes that the performance of BNP-FIRL(NEAT) is competitive to that of BNP-FIRL(mean), both of which are better than GPIRL and NEAT-IRL. The figure also establishes that in this setting, the use of neural networks alone (NEAT-IRL) outperforms GPIRL. The experiment in Figure 5.6 is also extended to observe performance in the presence of $9 - 16$ samples (calculated over 25 executions). This is shown in Figure 5.9. While the trends between GPIRL, BNP-FIRL(mean) and BNP-FIRL(NEAT) continue, limitation in performance of NEAT-IRL attributed to NEAT parameter settings is observed.

To compare BNP-FIRL(mean) and BNP-FIRL(NEAT) algorithms, the use of neural networks (BNP-FIRL(NEAT)) is tested for its better adaptability to non linear boundaries as compared to BNP-FIRL(mean). The algorithms are evaluated over various non-deterministic ($d = 0.7$) linear MDPs (varied by random seed initialization) to observe algorithm performance over different optimal policies for different MDPs. For representational tractability, n is limited to 4. Figure 5.10 depicts variation of misprediction score across various grids generated using specified seed values. The results are shown in Figure 5.10.

Overall, the performance of the two algorithms is indistinguishable. However, a closer look at specific seed values shows that BNP-FIRL(NEAT) performs noticeably better than BNP-FIRL(mean) for $seed = 7, 15, 24, 25$. The policies for the MDPs for these situations are shown in Figure 5.11 and Figure 5.12 respectively (arrows represent optimal direction of motion, if any). States with no arrows are goal states.

From these MDPs, it is unclear on whether the number of goal states discriminates

performance between the two algorithms. It is then hypothesized that BNP-FIRL(NEAT) performs better than BNP-FIRL(mean) in the presence of multiple goal states. To evaluate this, two MDPs ($n = 4$) are manually constructed with goals randomly placed in the MDP. This is done by associating an arbitrary reward value of 100 to those states. The differences in performances of the two algorithms are also compared for significance using two tailed t-test. The results are summarized in Table 5.1. $M_{BNP-FIRL(mean)}$ and $M_{BNP-FIRL(NEAT)}$ represent misprediction scores corresponding to BNP-FIRL(mean) and BNP-FIRL(NEAT) respectively. To overcome the fluctuation of numbers for averages over smaller number of runs (such as 25), the results are averaged over 1000 runs.

In the presence of a single goal state, BNP-FIRL(mean) significantly outperforms BNP-FIRL(NEAT). However, as the number of goals increases, BNP-FIRL(NEAT) eventually competes and later significantly outperforms BNP-FIRL(mean). From an MDP perspective, increasing the number of goals results in a more complex policy. This is because the state reward and hence state value surface has more than one optima. This is visually exemplified in Figure 5.13 and Figure 5.14. The hypothesis that the use of a neural network allows to learn more complicated reward structures is therefore confirmed.

The hypothesis is then tested on an $n = 16$ scale (with $N_S = 16$, $L_S = 4$ for correspondingly scaled input samples). However, experiments involving randomly placed goal states did not reflect the expected dominance of BNP-FIRL(NEAT) over BNP-FIRL(mean). To investigate this, MDPs on grids on an $n = 4$ scale were re-analysed for distribution of states where the optimal action is to take no action. On analysis, the number of such $no - action$ states does not show coherence with the number of goals states. This is reasoned as follows. In the case of an episodic task, it would be expected that the number and locations of goals correspond to the number and locations of $no - action$ states. However, the experimental setup used involves continuous tasks. In such a situation, the agent will continue to move in the MDP space even after reaching a state which would be a goal or

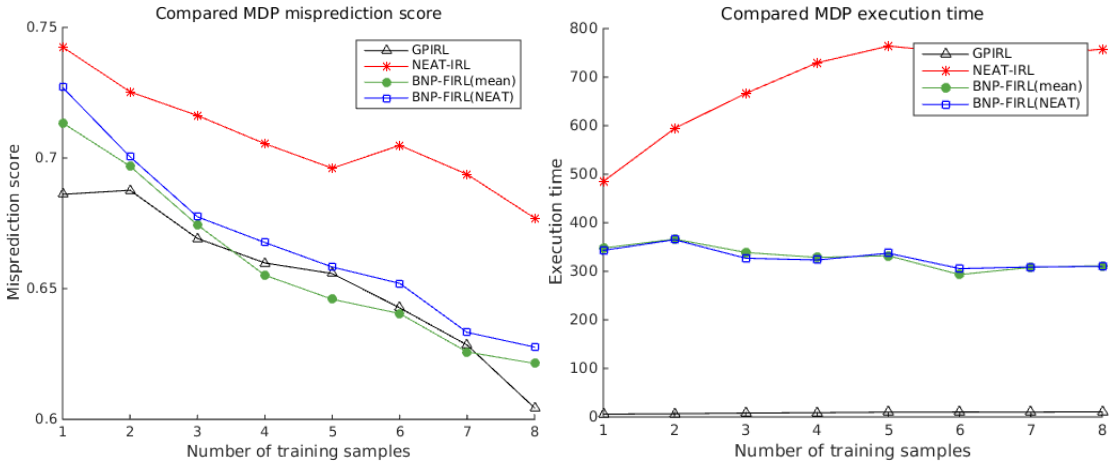

FIG. 5.4. Number of samples evaluation (standard MDP, $d = 0.7$)

Number of Goal States	$M_{BNP-FIRL(mean)}$	$M_{BNP-FIRL(NEAT)}$	$p - value$
1	0.2308	0.2545	3.8837 e-4
2	0.3292	0.3392	0.1870
3	0.4119	**0.3913**	**0.0063**
4	0.4954	**0.4674**	**3.9776 e-5**

Table 5.1. Performance on constructed MDPs

terminal state in the case of an episodic task. Based on these observations, four goals with arbitrary reward value 100 are placed at each corner of the grid. In an average over 100 executions, BNP-FIRL(NEAT) results in a significantly ($pvalue = 4.1819e06$) lower misprediction score (0.2672) as compared to BNP-FIRL(mean)(0.3072). The hypothesis that the use of a neural network allows to learn more complicated reward structures is therefore confirmed.

A conclusive performance hierarchy between these algorithms in a non-deterministic ($d = 0.7$) linear MDP experimental setting is then established as BNP-FIRL(NEAT) $>$ BNP-FIRL(mean) $>$ NEAT-IRL $>$ GPIRL. Additionally, a decision tree for algorithm selection for different MDP variations based on experimental results is presented in Figure 5.15. This figure is based on Figure 5.4, Figure 5.5, Figure 5.6 and Figure 5.7.

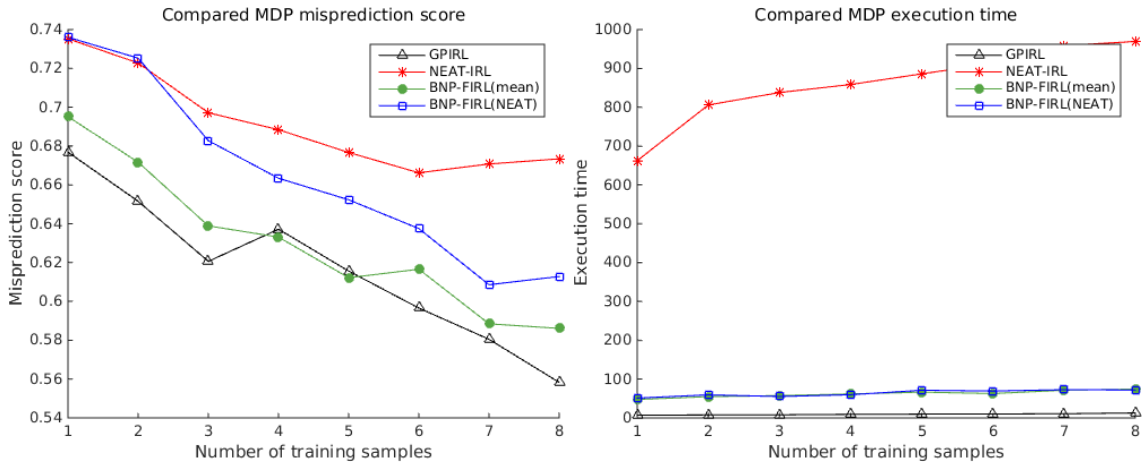

FIG. 5.5. Number of samples evaluation (standard MDP, $d = 1.0$)

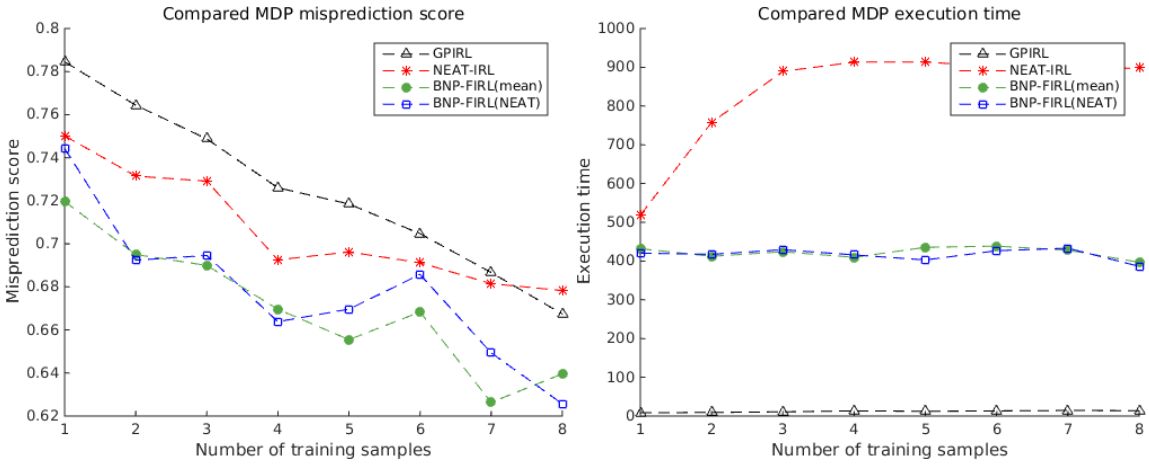

FIG. 5.6. Number of samples evaluation (linear MDP, $d = 0.7$)

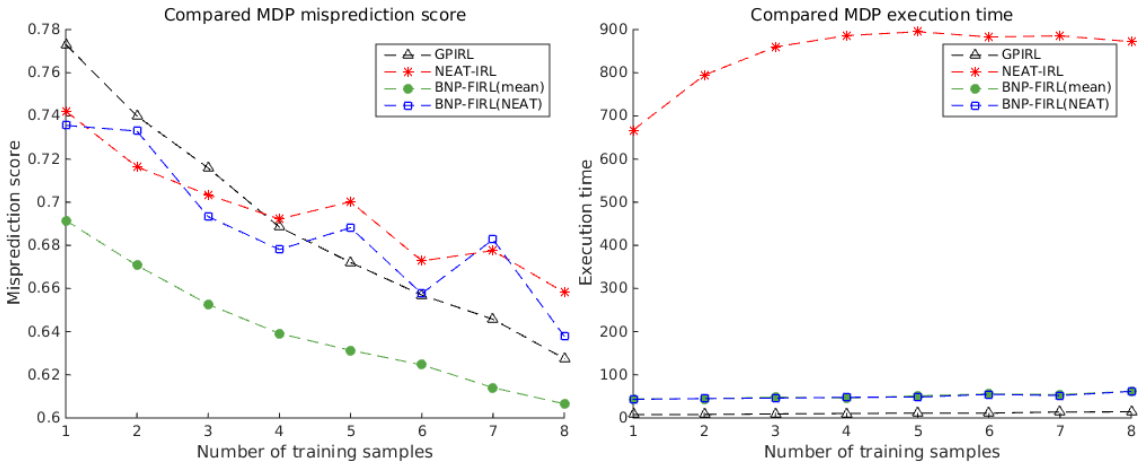

FIG. 5.7. Number of samples evaluation (linear MDP, $d = 1.0$)

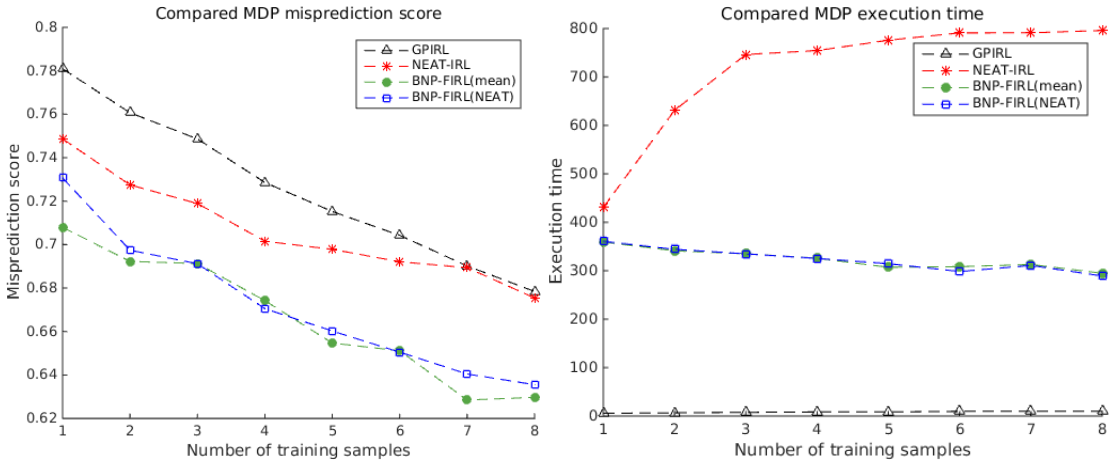

FIG. 5.8. Number of samples evaluation (linear MDP, $d = 0.7$, 100 executions)

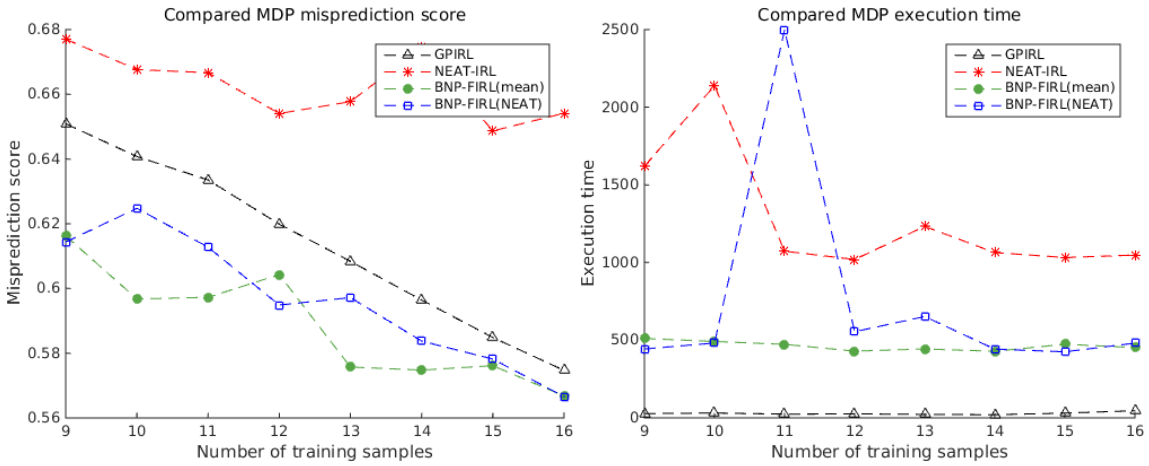

FIG. 5.9. Number of samples evaluation (linear MDP, $d = 0.7$, $9 - 16$ samples)

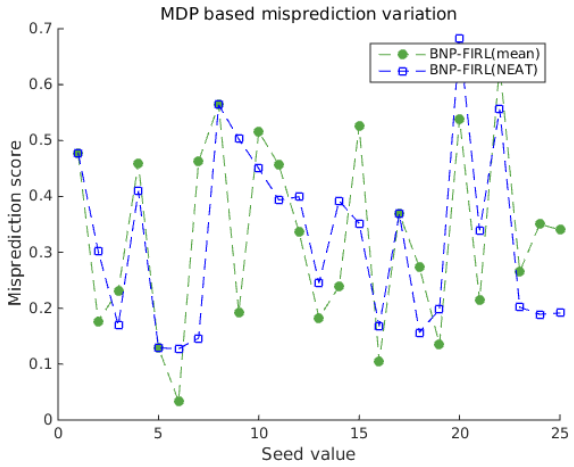

FIG. 5.10. MDP variation for BNP-FIRL(mean) and BNP-FIRL(NEAT)

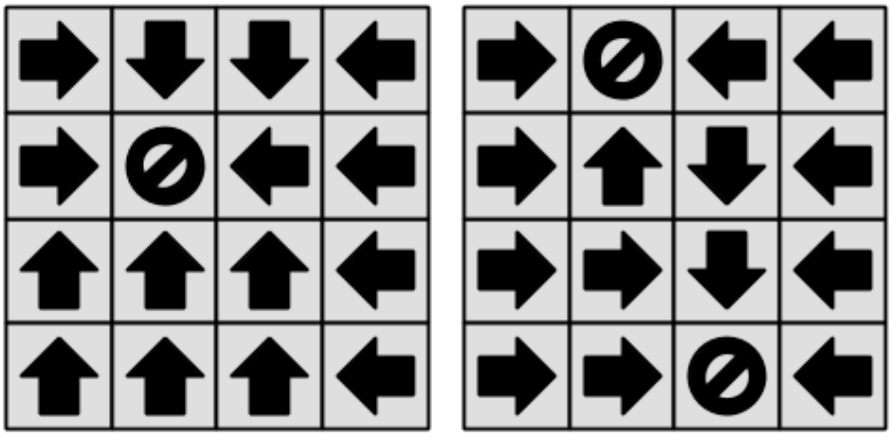

FIG. 5.11. MDP solutions (seeds 7, 15)

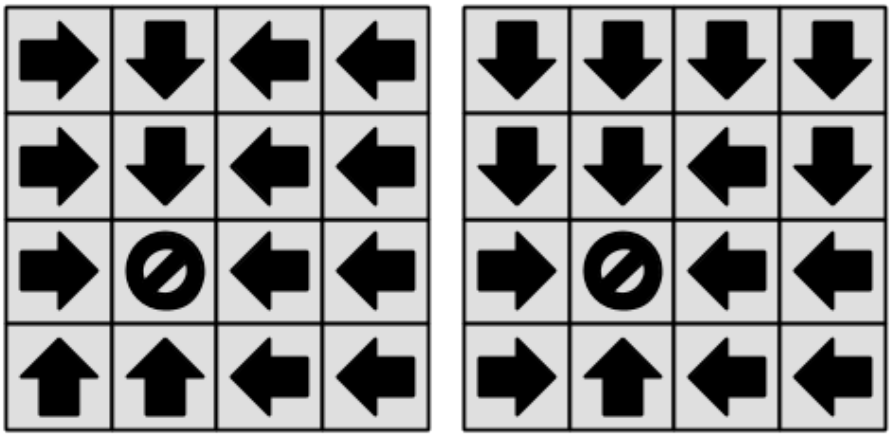

FIG. 5.12. MDP solutions (seeds 24, 25)

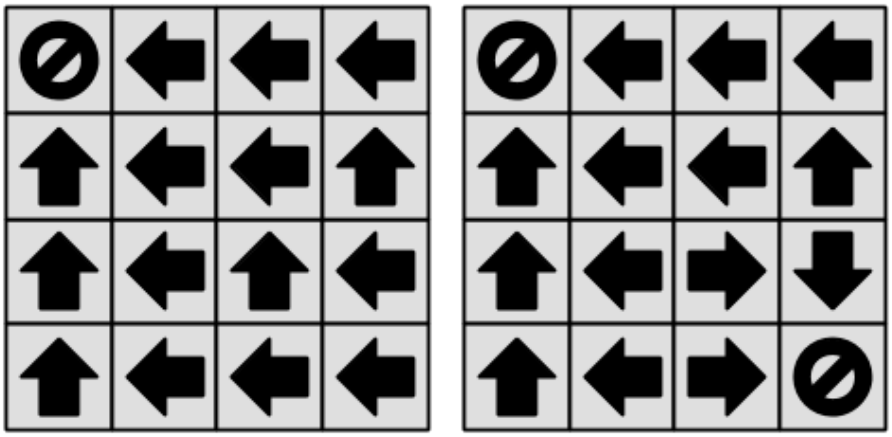

FIG. 5.13. Example MDP solutions (1 goal, 2 goals)

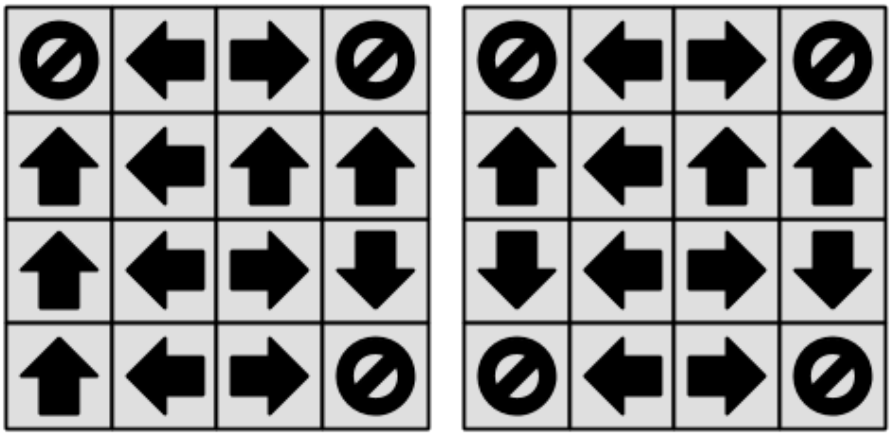

FIG. 5.14. Example MDP solutions (3 goal, 4 goals)

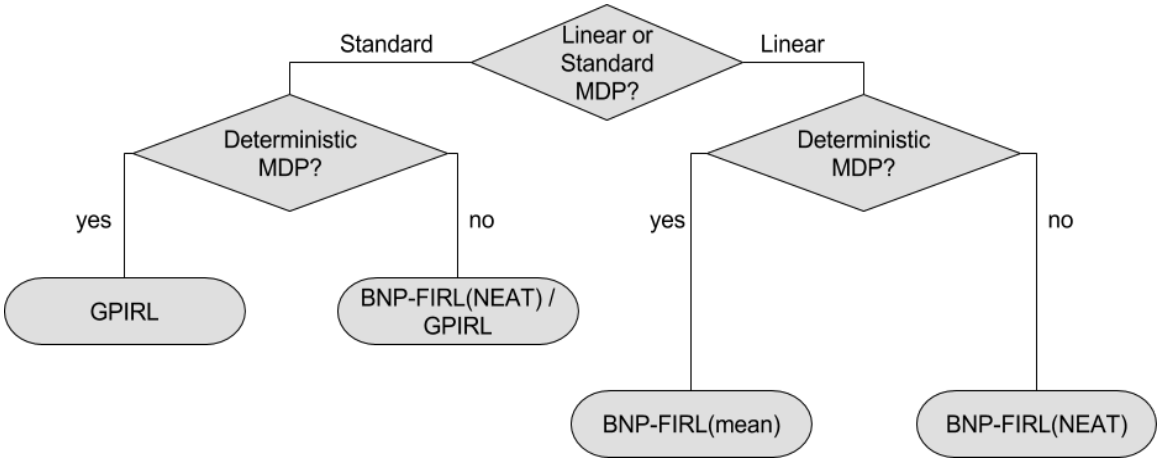

FIG. 5.15. Algorithm decision tree

Chapter 6

FUTURE WORK

These experiments conclude that algorithms using NEAT perform better on a non-deterministic linear MDP than BNP-FIRL(mean) and GPIRL (as in Figure 5.8). This is identified as useful considering that real world MDPs contain uncertainty in action caused by various sources of noise.

Given the competitive performance of BNP-FIRL(NEAT) and BNP-FIRL(mean), hospitable MDPs for BNP-FIRL(NEAT) are then evaluated and experiments highlight the possibility of MDPs with multiple goals being favourable (as in Figure 5.11 and 5.12). This hypothesis is examined and superior performance of BNP-FIRL(NEAT) is confirmed in cases of MDPs containing multiple goal states. BNP-FIRL(NEAT) is able to better estimate more complex reward structure (as in Table 5.1). A corresponding hierarchy of evaluated algorithms is then established with BNP-FIRL(NEAT) ranked the highest.

Additionally, NEAT parameters can be tuned to improve performance and time complexity for a given set of examples (as in Figure 5.2 and Figure 5.3). The current implementation of NEAT-IRL is also capable of greater time efficiency. Computations specific to each genome in a population can be parallelized. Further, NEAT-IRL policy prediction is currently done for all states. This can be limited to only demonstrated states, since that is what determines fitness.

In future work, BNP-FIRL(NEAT) may be integrated to multiple agent settings and may also be extended to incorporate a cost of sharing information (Miikkulainen *et al.* 2012).

Appendix A

SUPPLEMENTARY EVALUATIONS: NEAT-IRL, GPIRL AND FIRL

The purpose of this section is to include more exhaustive performance evaluations of NEAT-IRL, a purely neuroevolution based algorithm. The term purely is used to denote the lack of underlying preprocessing of feature information, as is done in the case of BNP-FIRL(NEAT). The experiments are conducted on a completely deterministic MDP ($d = 1.0$) to test learning capability without noise (random actions). As observed in other experiments, results of NEAT based algorithms are more pronounced on a linear MDP, which is the MDP type used for these experiments.

FIRL, GPIRL and NEAT-IRL are compared across three aspects of the demonstration and state space: N_S, L_S and n. For each of these evaluations, NEAT parameters remain arbitrarily set as $N_P = 50$ and $N_G = 50$. As in (Levine, Popovic, & Koltun 2011b), GPIRL is observed to be consistently better than FIRL.

Figure A.1 evaluates FIRL, GPIRL and NEAT-IRL over L_S ranging from 1 to 8. NEAT-IRL performs competitive to FIRL and GPIRL when sample lengths are lower (particularly $L_S = 1$). In particular, NEAT-IRL is better on a linear MDP and GPIRL is better on a standard MDP. However, when there is more information available, FIRL and GPIRL perform better than NEAT-IRL. This may be due to the capabilities of FIRL and GPIRL

when sufficient demonstration information is available, and limiting values of N_P and N_G (as justified in Figure A.4). FIRL misprediction score when tested on a linear MDP does not change much with example information quantity. This may be a limitation of internal optimization methods used for FIRL. It is evident from Figure A.1 that FIRL and GPIRL execution times are approximately independent of the length of the samples provided. The case is similar for NEAT-IRL after a threshold, but with a greater execution time. This may be due to potential overlap of examples, which effectively decreases demonstration space.

Dependency on N_S is evaluated by varying N_S from 1 to 8. The results are observed in Figure A.2. As in Figure A.1, NEAT-IRL performs better than FIRL and GPIRL on a linear MDP when less training samples are available. The reasoning for this performance remains the same. Execution time patterns are similar to Figure A.2. However, in the case of a smaller grid, the computational complexity of FIRL is briefly linear after a certain threshold (see Appendix C). This may be attributed to growth in its internal regression tree model dominating time complexity. It is interesting that corresponding increase in computational complexity in this case is much lower for NEAT-IRL than FIRL or GPIRL, which is attributed to NEAT-IRL matching all demonstrations in early generations, thereby terminating the algorithm. Increasing time complexity is attributed to fitting more data.

Further, the algorithms are evaluated over n (ranging from 4 to 16). This also means that the number of features increases from 6 to 30. Based on observations in Figure A.1, $L_S = 1$ is used to examine competitive performance of NEAT-IRL. A fraction of the state space is used for examples as $N_S = \lceil f \times n^2 \rceil$. Ceiling ensures a non empty set of examples. Using default parameters provided in (Levine, Popovic, & Koltun 2011a), $f = (16 \times 8)/(32 \times 32) = 0.125$. This is the value used for this evaluation. As in Figure A.3, NEAT-IRL performs better than GPIRL and FIRL in terms of misprediction score on a linear MDP for much of the evaluation. This does, however, incur a cost of non linear time complexity as opposed to what appears to be constant time complexity. This is attributed

to the increasing complexity of the neural network as the number of inputs (features) increases. The decrease in performance of NEAT-IRL for larger values of n may be attributed to unsuitability of the current values of N_P and N_G for the increasing sample space. This is experimentally verified by an observed decrease in performance in Figure A.4 with NEAT-IRL parameters set to be more limiting as $N_P = 5$ and $N_G = 5$ as opposed settings for Figure A.3. GPIRL performs almost strictly better than FIRL and NEAT-IRL when testing on a standard MDP. Execution times in Figure A.4 are more competitive because faster NEAT-IRL execution due to more limiting parameters. This also reflects an observable linearity and non linearity in time complexities for FIRL and GPIRL respectively, which is not as evident in Figure A.3.

In a final comparison, NEAT-IRL is tested for its better adaptability to non linear boundaries as compared to FIRL and GPIRL. For representational tractability, n is limited to 4. Figure A.5 depicts variation of misprediction score across various grids generated using specified seed values. The following use of a t-test refers to a two sample t-test. The scores of both GPIRL and NEAT-IRL are significantly better than those of FIRL for both linear and standard MDPs (t-test evaluates with $p - value$ less than 0.001). Similarly, scores of GPIRL are significantly better than those of NEAT-IRL for a standard MDP (t-test evaluates with $p - value$ less than 0.001). However, scores of NEAT-IRL are significantly better than those of GPIRL for a linear MDP (t-test evaluates with $p - value$ of 0.0378). NEAT-IRL performs almost as good as GPIRL for $seed = 5, 21$ and significantly better than GPIRL for $seed = 4, 19$. These particular seeds were not modified internally and are therefore valid for translation to corresponding optimal MDP policies. These policies for the MDPs for these two situations are shown in Figure A.6 and Figure A.7 respectively (arrows represent optimal direction of motion, if any). States with no arrows are goal states.

From these MDPs, it is unclear on whether location of the goal state discriminates performance of GPIRL and NEAT-IRL. It is then hypothesized that NEAT-IRL performs

better than GPIRL in the presence of multiple goal states. To evaluate this, two MDPs ($n = 4$) are manually constructed with 3 (arbitrary) scattered and clustered multiple reward structures as in Figure A.8. Blank states correspond to zero rewards and goal states have a reward of 100 (maximum reward set for the grid world in the toolkit). Goal states are scattered in the first MDP and clustered in the second. Performances of GPIRL and NEAT-IRL on these MDPs across various random seeds (used internally by the algorithms) are shown in Figure A.9. NEAT-IRL results in significantly better misprediction scores for a linear MDP than GPIRL in case of the scattered MDP (t-test evaluates with p-value of 0.0461. However, for the clustered MDP, performance difference between GPIRL and NEAT-IRL corresponds to p-value of 0.4623 on a t-test.

To evaluate this hypothesis further, 25 MDPs ($n = 4$) are generated with 2 and 3 (selected arbitrarily) goal states. GPIRL and NEAT-IRL are trained on standard and linear MDPs and are evaluated on a linear MDP (in view of better performance of NEAT-IRL than GPIRL for this metric). The findings are summarized in Table A.1. M_{GPIRL} and $M_{NEAT-IRL}$ refer to average misprediction scores for GPIRL and NEAT-IRL respectively. There is not significant improvement by NEAT-IRL over GPIRL when there are 3 goals. This is associated with there being less space in the MDP to scatter goals in proportion to the number of goals. This is affirmed by the improved performance of NEAT-IRL over GPIRL in case of 2 goals. NEAT-IRL is therefore established to be better suited than GPIRL for an MDP containing multiple sparsely located goal states. Misprediction scores when testing on a standard MDP favour GPIRL.

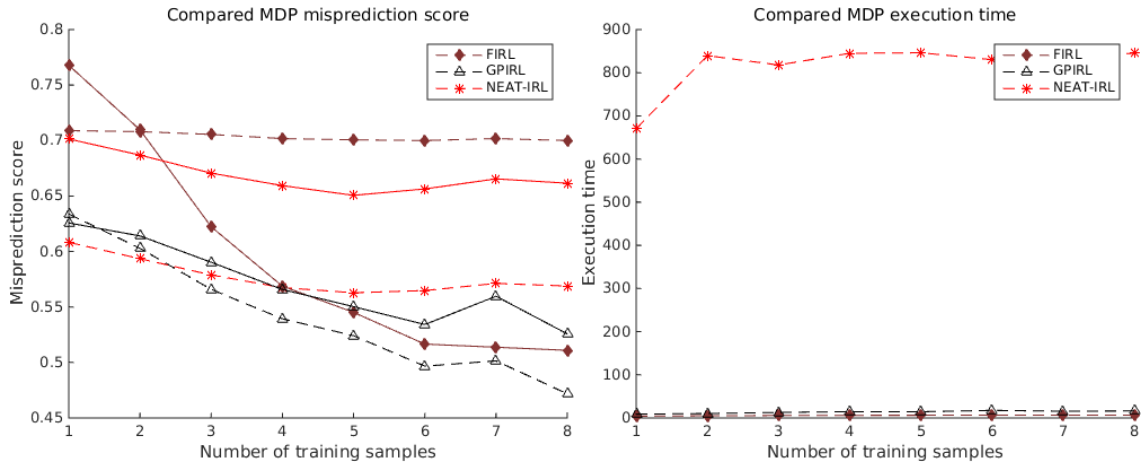

FIG. A.1. Sample length evaluation (linear MDP, $d = 1.0$)

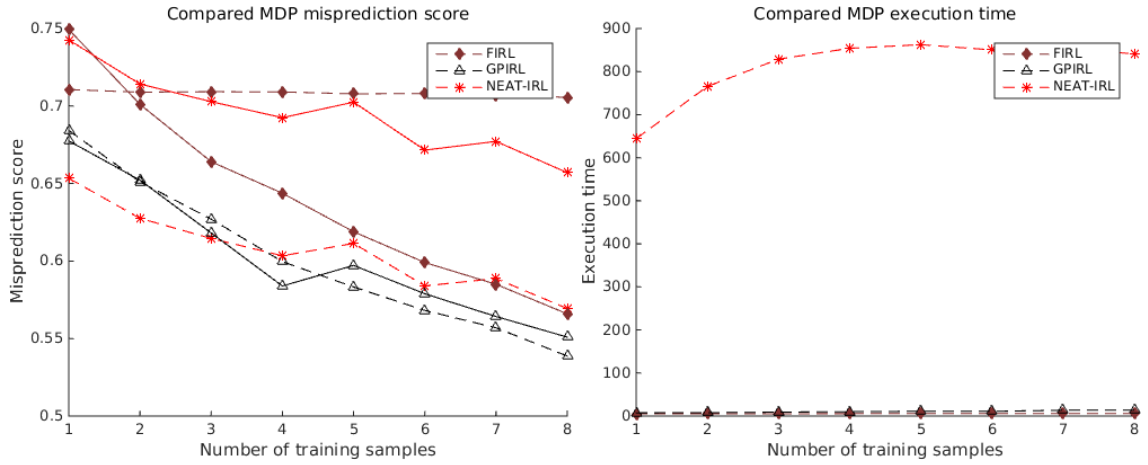

FIG. A.2. Number of samples evaluation (linear MDP, $d = 1.0$)

Number of Goal States	Training MDP	M_{GPIRL}	$M_{NEAT-IRL}$	$p - value$
2	Standard	0.4620	0.3731	**0.0077**
	Linear	0.4786	0.4027	**0.0150**
3	Standard	0.5055	0.5001	0.8510
	Linear	0.5057	0.4801	0.4043

Table A.1. Performance on manually constructed MDPs

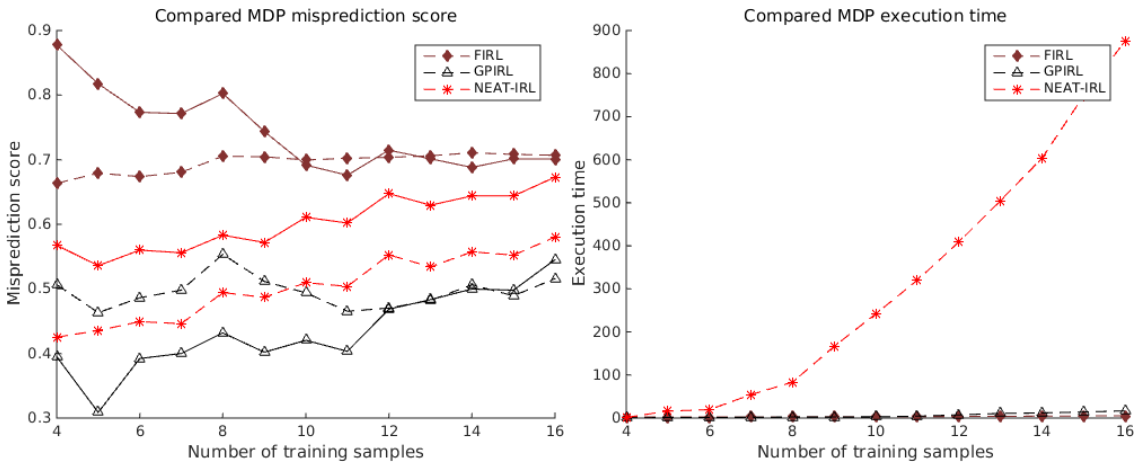

FIG. A.3. Grid size evaluation (linear MDP, $d = 1.0$)

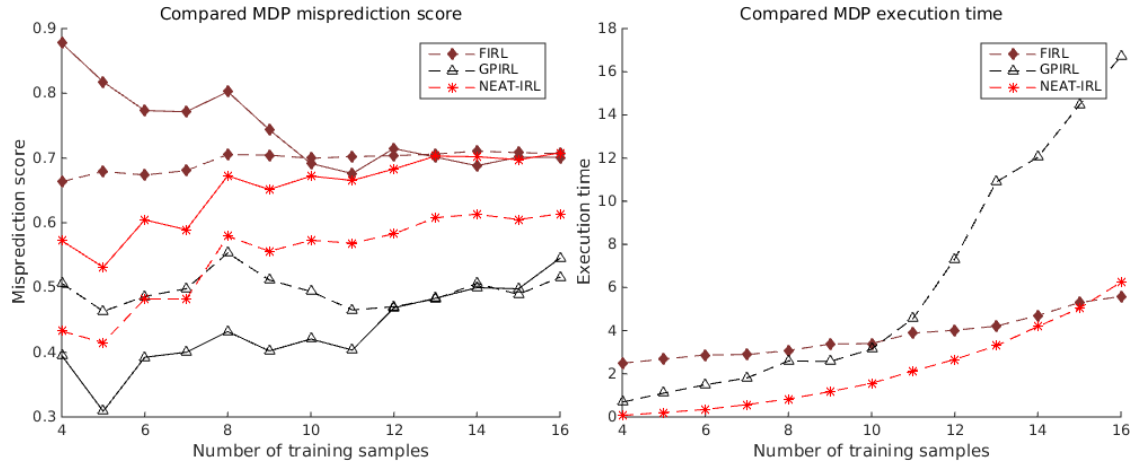

FIG. A.4. Parameter limitations ($N_P = 5$, $N_G = 5$)

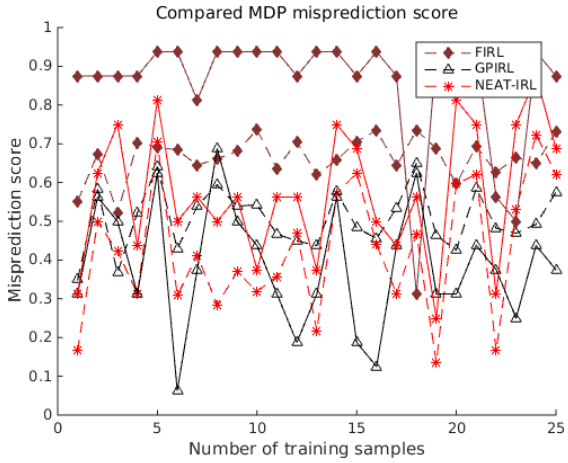

FIG. A.5. MDP variation (linear MDP, $d = 1.0$)

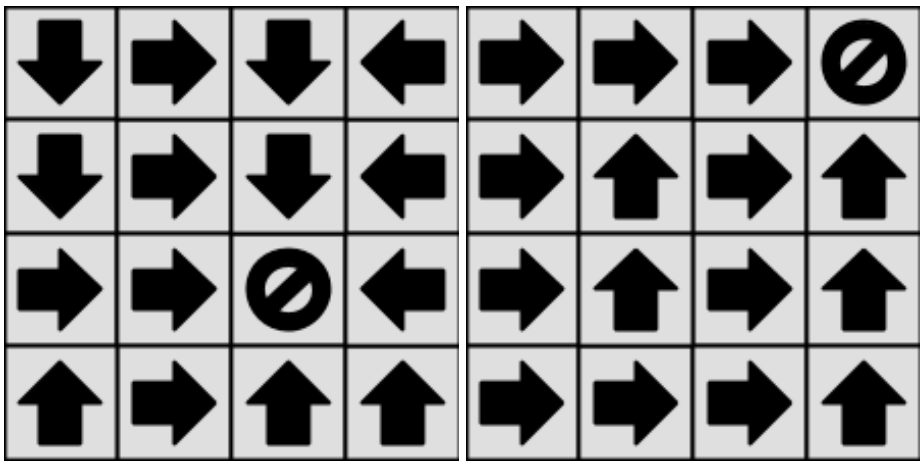

FIG. A.6. MDP solutions (seeds 5, 21)

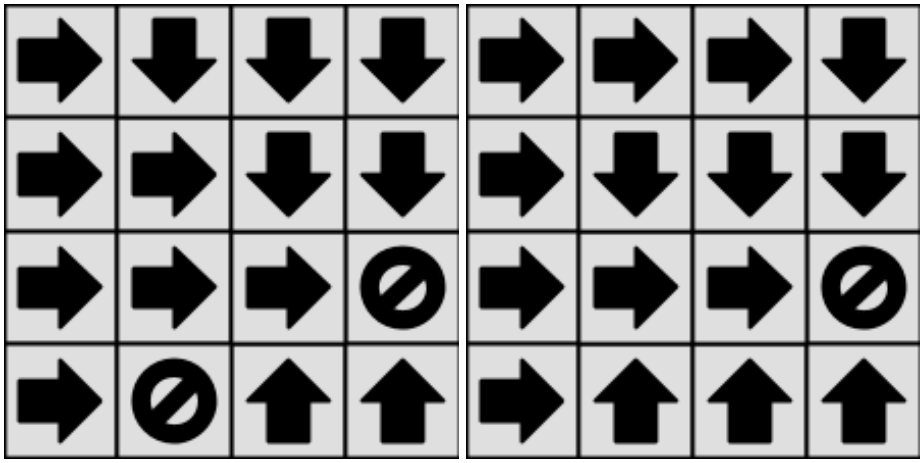

FIG. A.7. MDP solutions (seeds 4, 19)

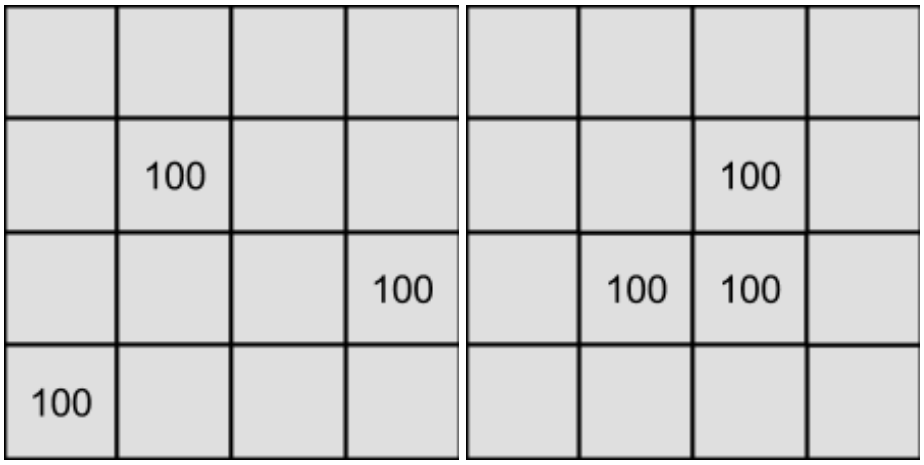

FIG. A.8. Manually constructed MDPs

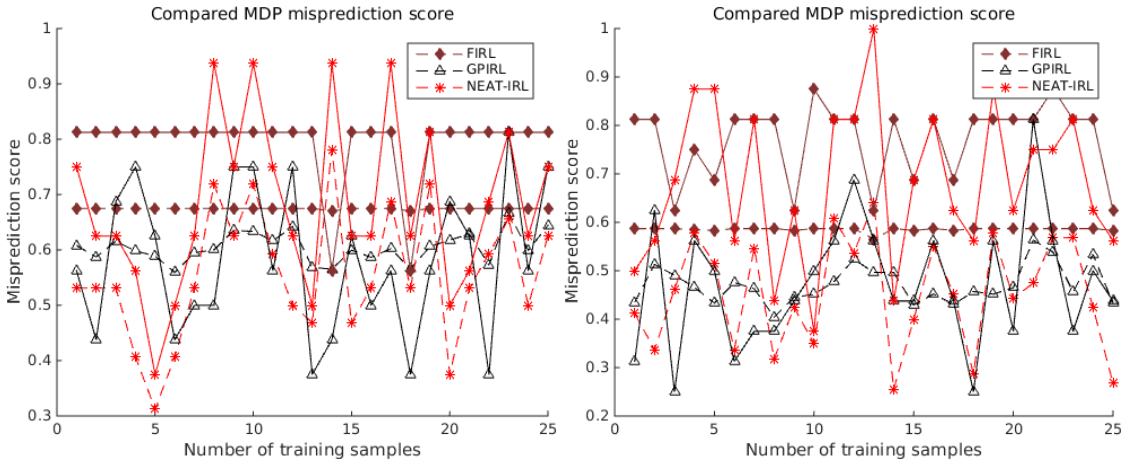

FIG. A.9. Performance on manually constructed MDPs (linear MDP, $d = 1.0$)

Appendix B

EXTENDED EVALUATIONS ($N = 16$): NEAT-IRL, GPIRL AND FIRL

In contrast to the figures included in Appendix A, the following are results obtained when the IRL models are trained on a standard MDP. Figure B.2 is an exception to this and generated by IRL models trained using a linear MDP. These figures serve to complement figures in Appendix A and provide a complete experimental result over standard and linear MDPs.

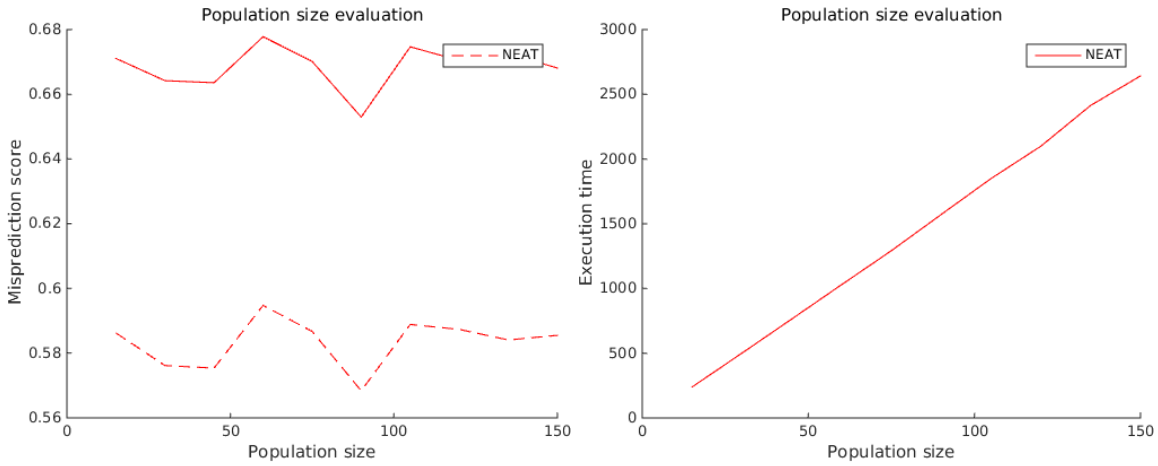

FIG. B.1. NEAT-IRL population size evaluation (standard MDP, $d = 1.0$)

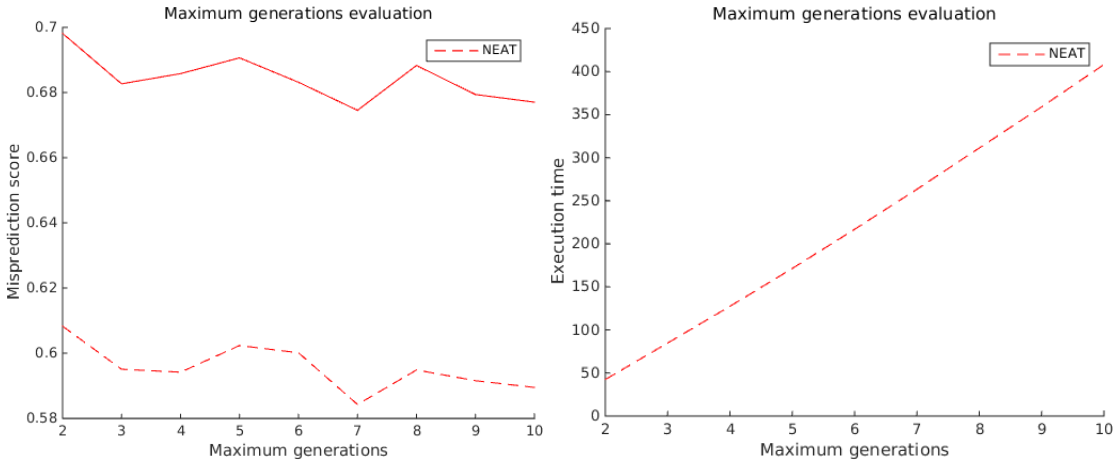

FIG. B.2. NEAT-IRL maximum generations evaluation (linear MDP, $d = 1.0$)

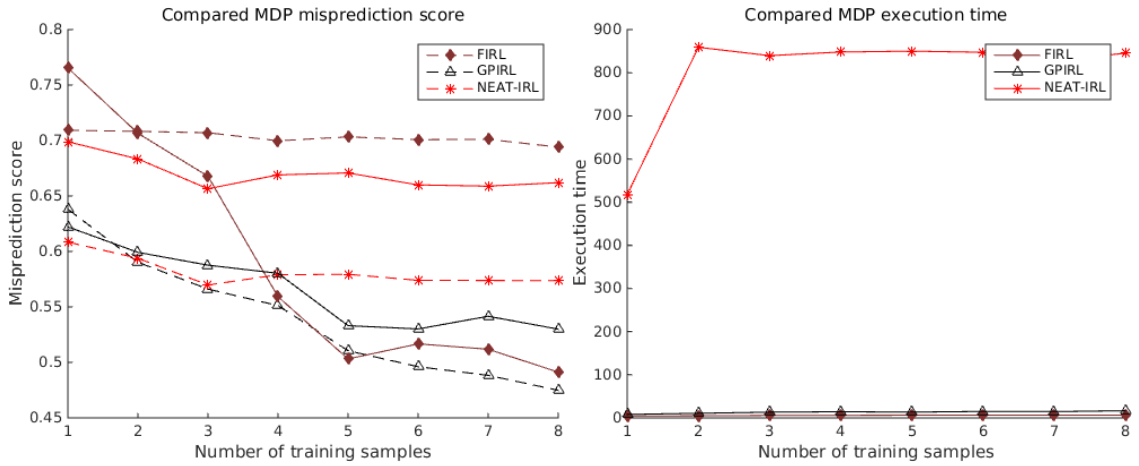

FIG. B.3. Sample length evaluation (standard MDP, $d = 1.0$)

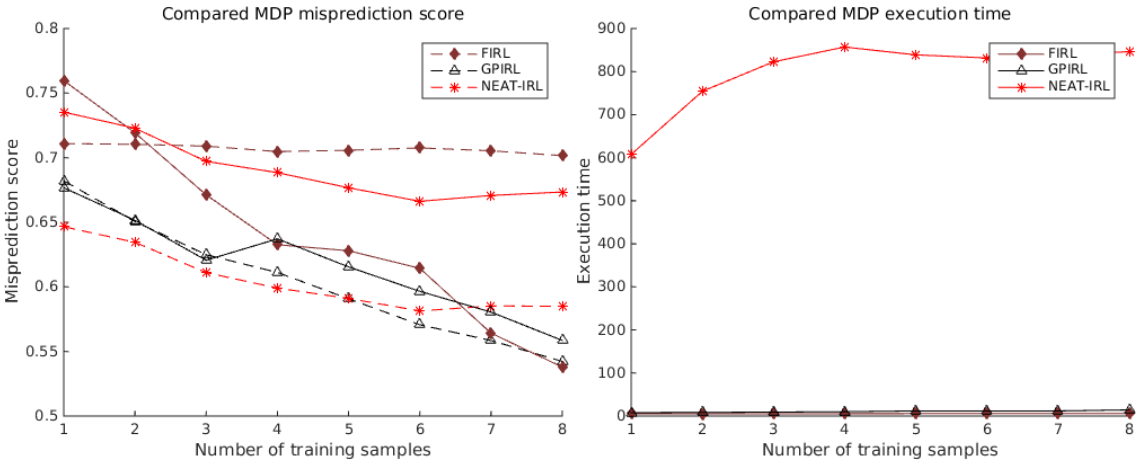

FIG. B.4. Number of samples evaluation (standard MDP, $d = 1.0$)

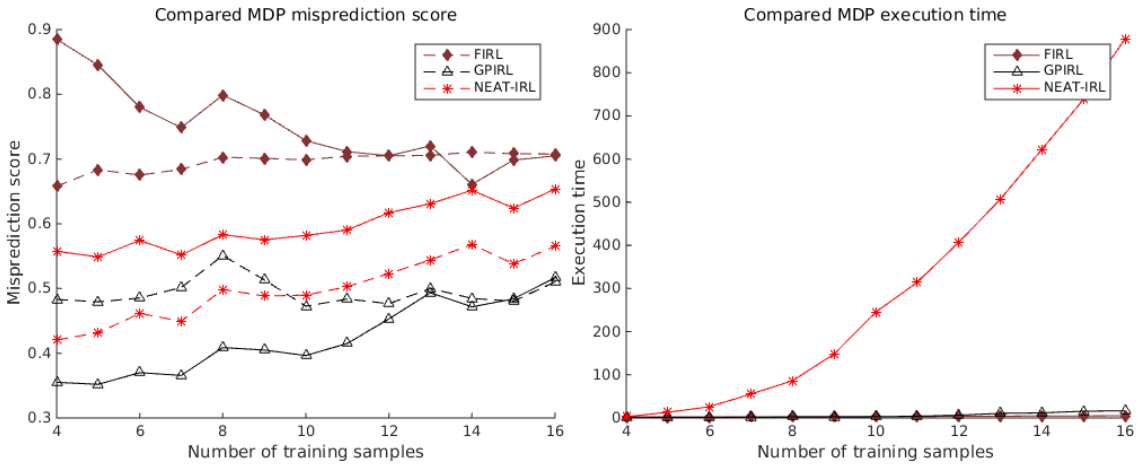

FIG. B.5. Grid size evaluation (standard MDP, $d = 1.0$)

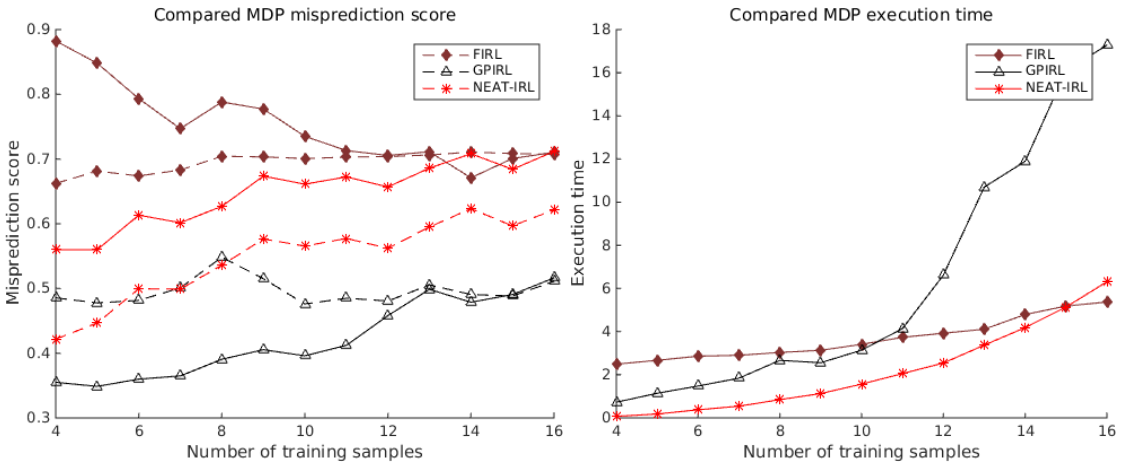

FIG. B.6. NEAT-IRL parameter limitations (standard MDP, $d = 1.0$)

Appendix C

SUPPLEMENTARY EVALUATIONS ($N = 4$): NEAT-IRL, GPIRL AND FIRL

This section contains results for experiments originally conducted for grid with $n = 16$, repeated on a grid with $n = 4$. These exclude results in Appendix D. This is done to evaluate the performance of NEAT-IRL parameters ($N_P = 50$, $N_G = 50$) on a less complex MDP. Figures C.1, C.2, C.3 and C.4 correspond to training on a linear MDP. Figures C.5, C.6, C.7, and C.8 correspond to training on a standard MDP. Performance stagnation is observed in Figures C.1, C.2, C.5 and C.6. This is attributed to NEAT-IRL parameters being larger than necessary, leading to early algorithm termination.

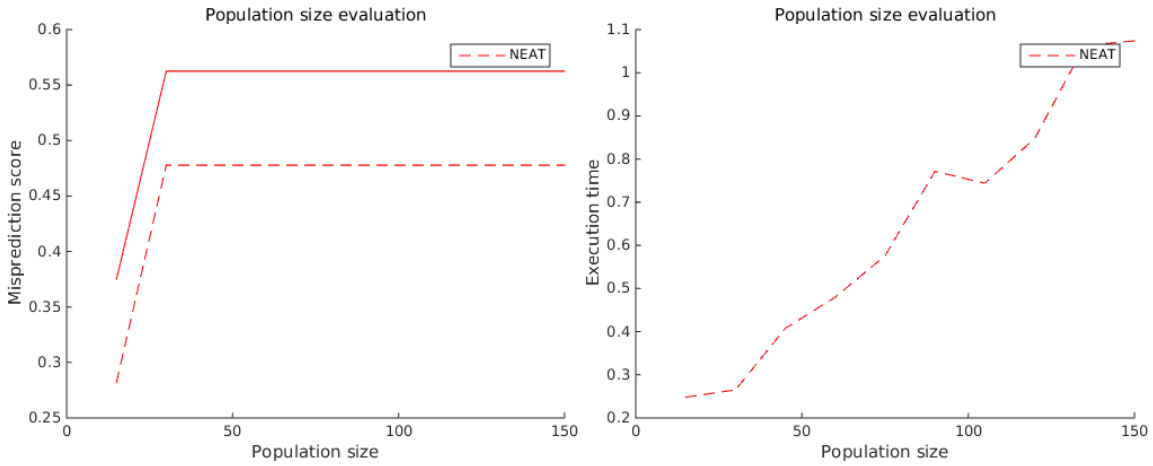

FIG. C.1. NEAT-IRL population size evaluation (linear MDP, $d = 1.0$)

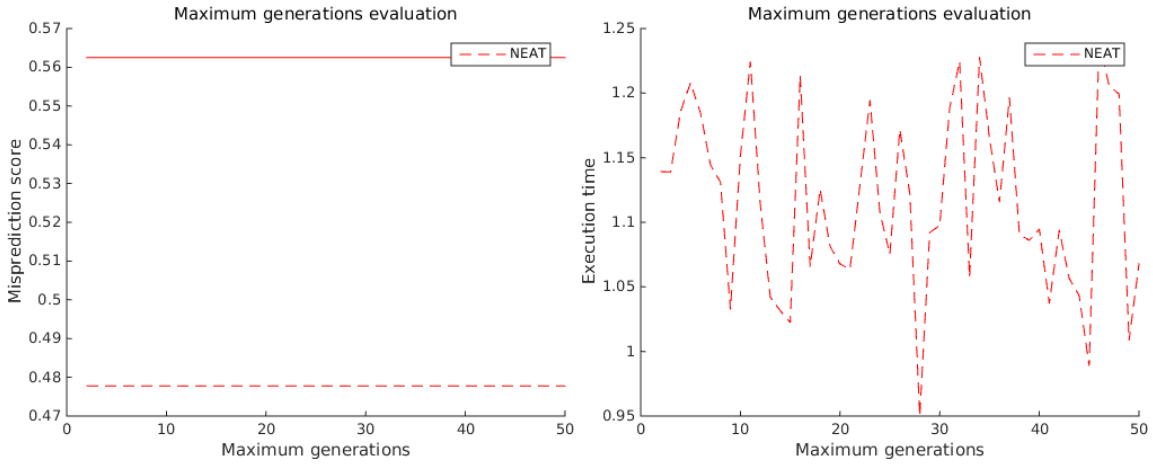

FIG. C.2. NEAT-IRL maximum generations evaluation (linear MDP, $d = 1.0$)

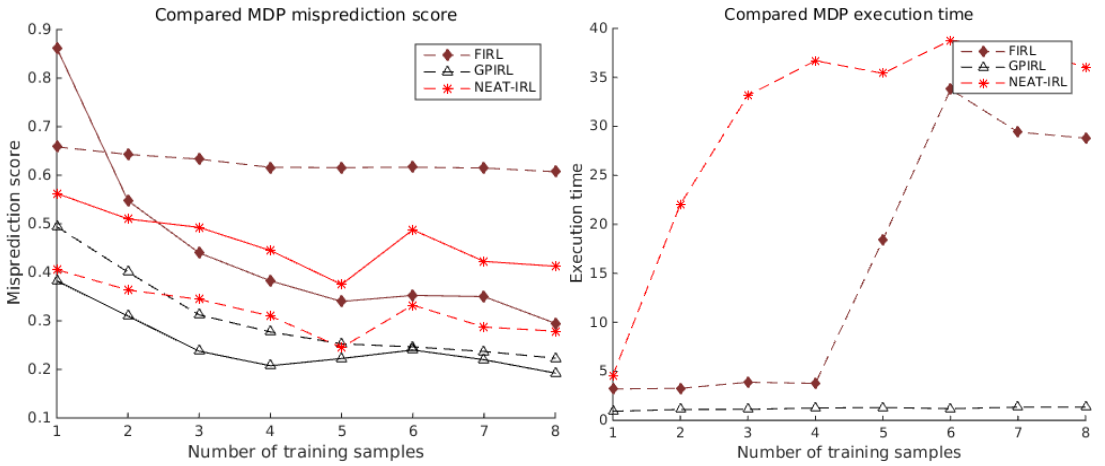

FIG. C.3. Sample length evaluation (linear MDP, $d = 1.0$)

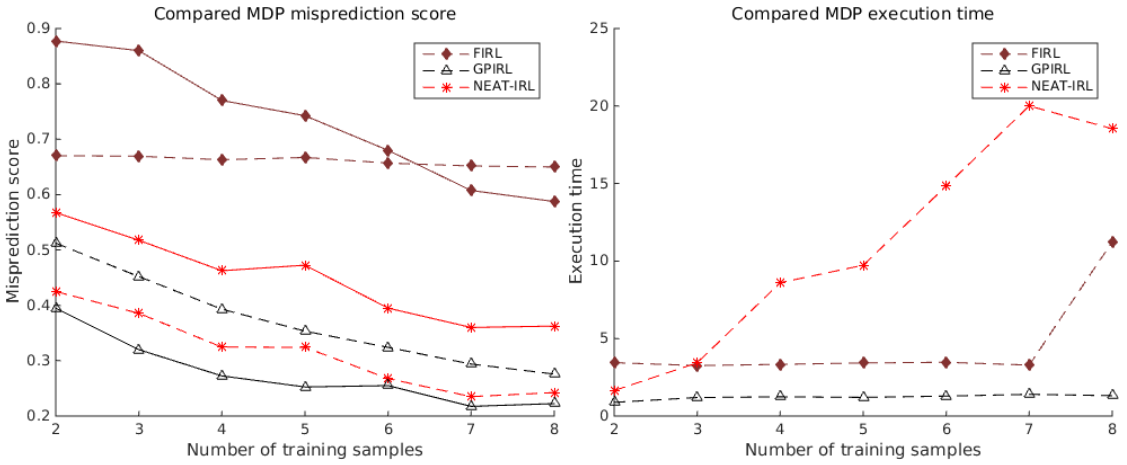

FIG. C.4. Number of samples evaluation (linear MDP, $d = 1.0$)

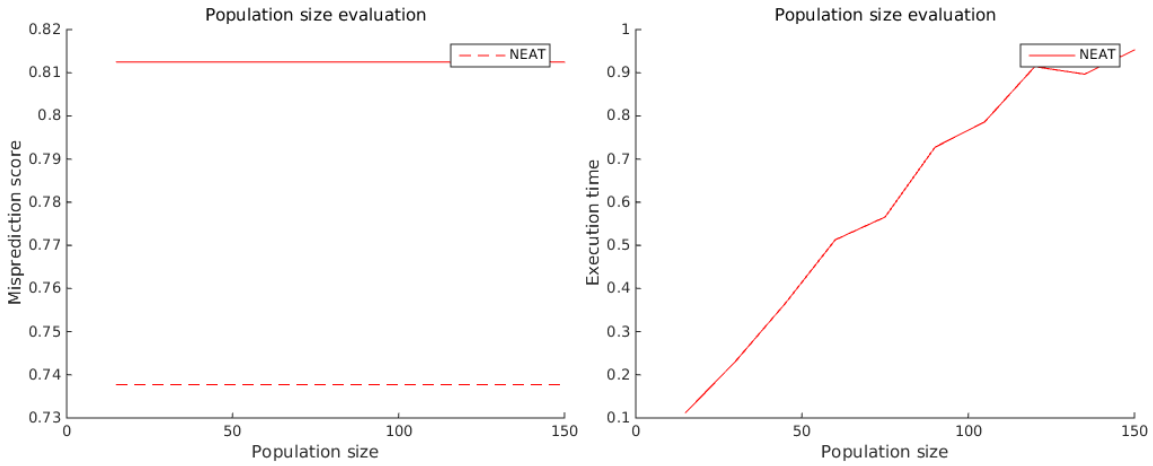

FIG. C.5. NEAT-IRL population size evaluation (standard MDP, $d = 1.0$)

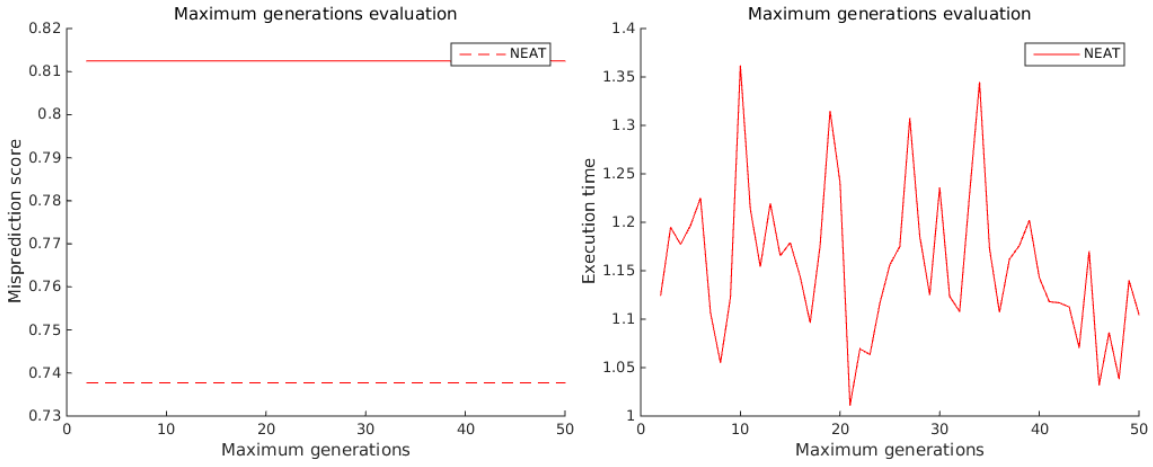

FIG. C.6. NEAT-IRL maximum generations evaluation (standard MDP, $d = 1.0$)

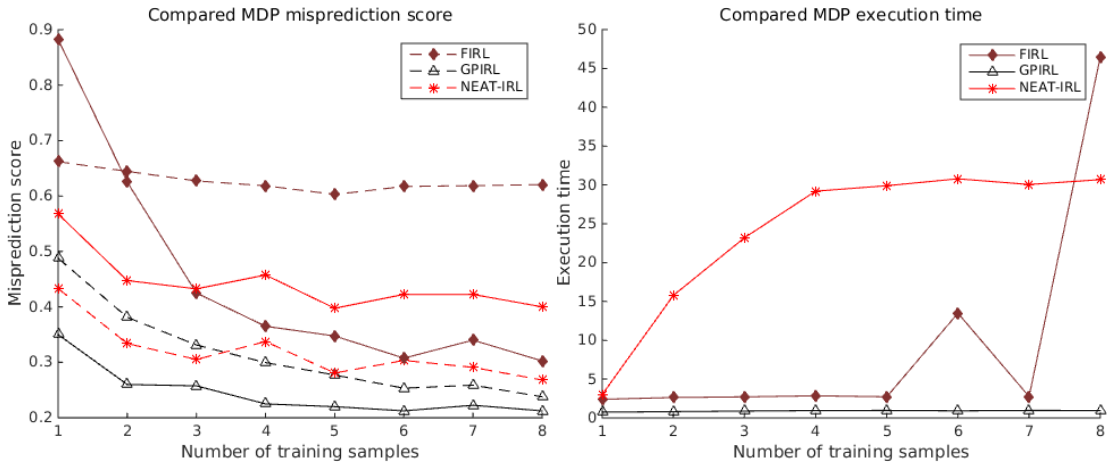

FIG. C.7. Sample length evaluation (standard MDP, $d = 1.0$)

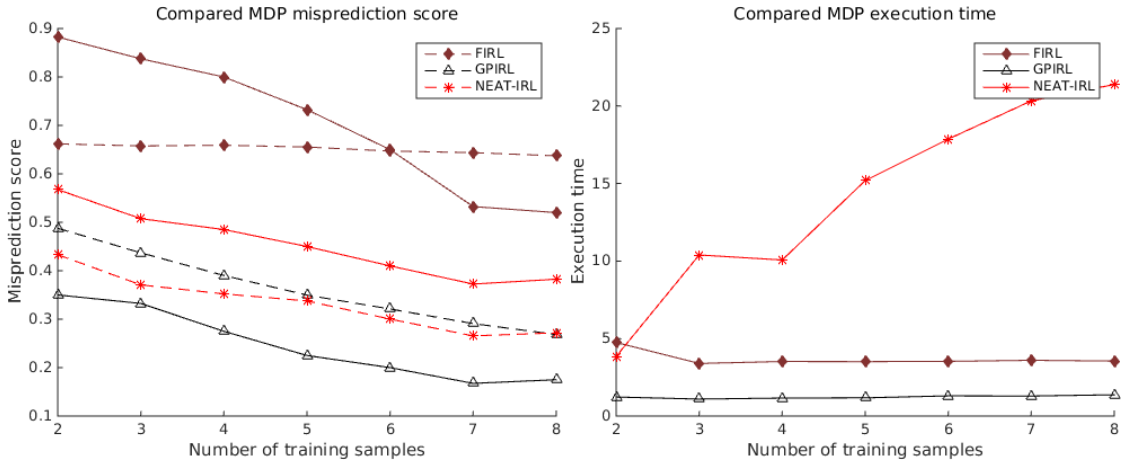

FIG. C.8. Number of samples evaluation (standard MDP, $d = 1.0$)

Appendix D

EXTENDED EVALUATIONS ($N = 4$): NEAT-IRL, GPIRL AND FIRL

This section completes Appendix C by providing results obtained on MDP analysis experiments when the IRL models are trained on a standard MDP.

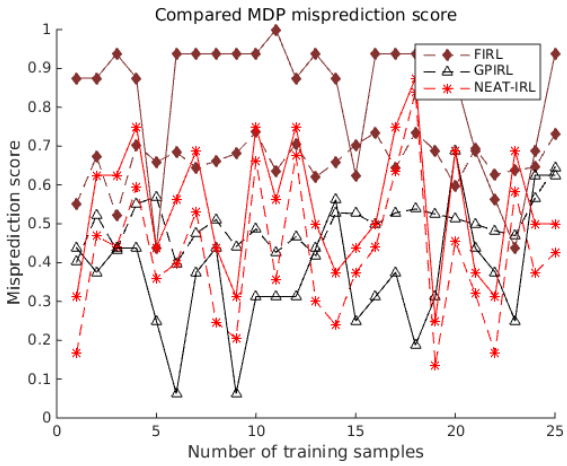

FIG. D.1. MDP variation (standard MDP, $d = 1.0$)

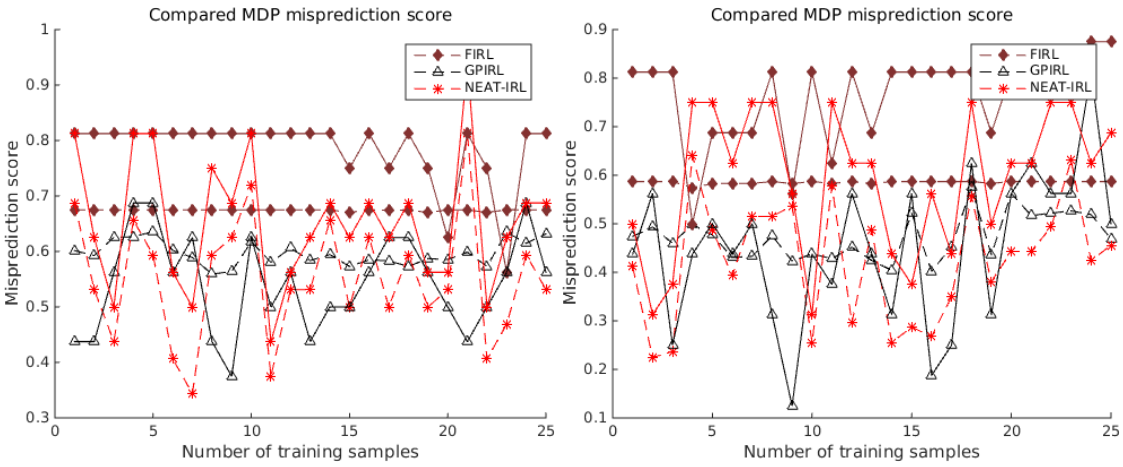

FIG. D.2. Performance on manually constructed MDPs (standard MDP, $d = 1.0$)

REFERENCES

[1] Abbeel, P., and Ng, A. Y. 2004. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, 1. ACM.

[2] Abbeel, P. 2012. Inverse reinforcement learning. `http://www.cs.berkeley.edu/~pabbeel/cs287-fa12/slides/inverseRL.pdf`.

[3] Angeline, P. J.; Saunders, G. M.; and Pollack, J. B. 1994. An evolutionary algorithm that constructs recurrent neural networks. *Neural Networks, IEEE Transactions on* 5(1):54–65.

[4] Banzhaf, W.; Nordin, P.; Keller, R. E.; and Francone, F. D. 1998. *Genetic programming: an introduction*, volume 1. Morgan Kaufmann San Francisco.

[5] Barto, A. G. 1998. *Reinforcement learning: An introduction*. MIT press.

[6] Bengio, Y. 2009. Learning deep architectures for ai. *Foundations and trends® in Machine Learning* 2(1):1–127.

[7] Caruana, R., and Niculescu-Mizil, A. 2006. An empirical comparison of supervised learning algorithms. In *Proceedings of the 23rd international conference on Machine learning*, 161–168. ACM.

[8] Chervenski, P. 2012. Multineat. `http://multineat.com/`.

[9] Choi, J., and Kim, K.-E. 2011. Map inference for bayesian inverse reinforcement learning. In *Advances in Neural Information Processing Systems*, 1989–1997.

[10] Choi, J., and Kim, K.-E. 2013a. Bayesian nonparametric feature construction for inverse reinforcement learning. In *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, 1287–1293. AAAI Press.

[11] Choi, J., and Kim, K.-E. 2013b. Irl toolkit. `http://ailab.kaist.ac.kr/codes/bayesian-nonparametric-feature-construction-for-irl`.

[12] Deisenroth, M. P.; Rasmussen, C. E.; and Peters, J. 2009. Gaussian process dynamic programming. *Neurocomputing* 72(7):1508–1524.

[13] Deng, L., and Yu, D. 2014. Deep learning: methods and applications. *Foundations and Trends in Signal Processing* 7(3–4):197–387.

[14] Engel, Y.; Mannor, S.; and Meir, R. 2005. Reinforcement learning with gaussian processes. In *Proceedings of the 22nd international conference on Machine learning*, 201–208. ACM.

[15] Ghahramani, Z., and Griffiths, T. L. 2005. Infinite latent feature models and the indian buffet process. In *Advances in neural information processing systems*, 475–482.

[16] Gruau, F., et al. 1994. Neural network synthesis using cellular encoding and the genetic algorithm.

[17] Hahn, J., and Zoubir, A. M. 2015. Inverse reinforcement learning using expectation maximization in mixture models. In *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, 3721–3725. IEEE.

[18] Haykin, S., and Network, N. 2004. A comprehensive foundation. *Neural Networks* 2(2004).

[19] Karpov, I. V.; Valsalam, V. K.; and Miikkulainen, R. 2011. Human-assisted neuroevolution through shaping, advice and examples. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, 371–378. ACM.

[20] Kassahun, Y., and Sommer, G. 2005. Efficient reinforcement learning through evolutionary acquisition of neural topologies. In *ESANN*, 259–266.

[21] Koza, J. R. 1992. *Genetic programming: on the programming of computers by means of natural selection*, volume 1. MIT press.

[22] Kullback, S., and Leibler, R. A. 1951. On information and sufficiency. *The annals of mathematical statistics* 79–86.

[23] Levine, S.; Popovic, Z.; and Koltun, V. 2010. Feature construction for inverse reinforcement learning. In *Advances in Neural Information Processing Systems*, 1342–1350.

[24] Levine, S.; Popovic, Z.; and Koltun, V. 2011a. Irl toolkit. `http://graphics.stanford.edu/projects/gpirl/irl_toolkit.zip`.

[25] Levine, S.; Popovic, Z.; and Koltun, V. 2011b. Nonlinear inverse reinforcement learning with gaussian processes. In *Advances in Neural Information Processing Systems*, 19–27.

[26] Lilley, M., and Frean, M. 2005. Neural networks: a replacement for gaussian processes? In *Intelligent Data Engineering and Automated Learning-IDEAL 2005*. Springer. 195–202.

[27] Mayr, C. 2003. Matlab neat. `http://nn.cs.utexas.edu/?neatmatlab`.

[28] Michini, B., and How, J. P. 2012. Bayesian nonparametric inverse reinforcement learning. In *Machine Learning and Knowledge Discovery in Databases*. Springer. 148–163.

[29] Michini, B.; Walsh, T. J.; Agha-Mohammadi, A.-A.; and How, J. P. 2015. Bayesian nonparametric reward learning from demonstration. *Robotics, IEEE Transactions on* 31(2):369–386.

[30] Miikkulainen, R.; Feasley, E.; Johnson, L.; Karpov, I.; Rajagopalan, P.; Rawal, A.; and Tansey, W. 2012. Multiagent learning through neuroevolution. In *Advances in Computational Intelligence*. Springer. 24–46.

[31] Neal, R. M. 1995. *Bayesian learning for neural networks*. Ph.D. Dissertation, University of Toronto.

[32] Neal, R. M. 1996. Priors for infinite networks. In *Bayesian Learning for Neural Networks*. Springer. 29–53.

[33] Ng, A. Y.; Russell, S. J.; et al. 2000. Algorithms for inverse reinforcement learning. In *Icml*, 663–670.

[34] Puterman, M. L. 2014. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.

[35] Ramachandran, D., and Amir, E. 2007. Bayesian inverse reinforcement learning. *Urbana* 51:61801.

[36] Rasmussen, C. E.; Kuss, M.; et al. 2003. Gaussian processes in reinforcement learning. In *NIPS*, volume 4, 1.

[37] Rasmussen, C. E. 2006. Gaussian processes for machine learning.

[38] Ratliff, N. D.; Bagnell, J. A.; and Zinkevich, M. A. 2006. Maximum margin planning. In *Proceedings of the 23rd international conference on Machine learning*, 729–736. ACM.

[39] Ratliff, N.; Bradley, D.; Bagnell, J. A.; and Chestnutt, J. 2007. Boosting structured prediction for imitation learning. *Robotics Institute* 54.

[40] Ratliff, N. D.; Silver, D.; and Bagnell, J. A. 2009. Learning to search: Functional gradient techniques for imitation learning. *Autonomous Robots* 27(1):25–53.

[41] Rempis, C. W. 2012. Evolving complex neuro-controllers with interactively constrained neuro-evolution.

[42] Sher, G. I. 2012. *Handbook of neuroevolution through Erlang*. Springer Science & Business Media.

[43] Siebel, N. T., and Sommer, G. 2007. Evolutionary reinforcement learning of artificial neural networks. *International Journal of Hybrid Intelligent Systems* 4(3):171–183.

[44] Stanley, K. O., and Miikkulainen, R. 2002. Evolving neural networks through augmenting topologies. *Evolutionary computation* 10(2):99–127.

[45] Stanley, K. O.; Bryant, B. D.; and Miikkulainen, R. 2005a. Evolving neural network agents in the nero video game. *Proceedings of the IEEE* 182–189.

[46] Stanley, K. O.; Bryant, B. D.; and Miikkulainen, R. 2005b. Real-time neuroevolution in the nero video game. *Evolutionary Computation, IEEE Transactions on* 9(6):653–668.

[47] Stanley, K. 2014. The neuroevolution of augmenting topologies (neat) users page. `http://www.cs.ucf.edu/~kstanley/neat.html`.

[48] Sutton, R. S., and Barto, A. G. 1998. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge.

[49] Syed, U., and Schapire, R. E. 2007. A game-theoretic approach to apprenticeship learning. In *Advances in neural information processing systems*, 1449–1456.

[50] Syed, U.; Bowling, M.; and Schapire, R. E. 2008. Apprenticeship learning using linear programming. In *Proceedings of the 25th international conference on Machine learning*, 1032–1039. ACM.

[51] Todorov, E. 2006. Linearly-solvable markov decision problems. In *Advances in neural information processing systems*, 1369–1376.

[52] Tufts University, M. M. 2015. Comp 131: Artificial intelligence. `http://www.cs.tufts.edu/comp/131`.

[53] Vroman, M. C. 2014. *Maximum likelihood inverse reinforcement learning*. Ph.D. Dissertation, Rutgers University-Graduate School-New Brunswick.

[54] Whiteson, S.; Stone, P.; Stanley, K. O.; Miikkulainen, R.; and Kohl, N. 2005. Automatic feature selection in neuroevolution. In *Proceedings of the 7th annual conference on Genetic and evolutionary computation*, 1225–1232. ACM.

[55] Wikipedia. 2015. Artificial neural network — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/Artificial_neural_network`.

[56] Williams, C. K., and Rasmussen, C. E. 2006. Gaussian processes for machine learning. *the MIT Press* 2(3):4.

[57] Wulfmeier, M.; Ondruska, P.; and Posner, I. 2015. Deep inverse reinforcement learning. *arXiv preprint arXiv:1507.04888*.

[58] Yao, X., and Liu, Y. 1997. A new evolutionary system for evolving artificial neural networks. *Neural Networks, IEEE Transactions on* 8(3):694–713.

[59] Yong, C. H.; Stanley, K. O.; Miikkulainen, R.; and Karpov, I. 2006. Incorporating advice into neuroevolution of adaptive agents. In *AIIDE*, 98–104.

[60] Ziebart, B. D.; Maas, A. L.; Bagnell, J. A.; and Dey, A. K. 2008. Maximum entropy inverse reinforcement learning. In *AAAI*, 1433–1438.

[61] Ziebart, B. D. 2010. Modeling purposeful adaptive behavior with the principle of maximum causal entropy.