Parallel Bayesian Model Learning

Enda Barrett  
College of Engineering and Informatics  
National University of Ireland, Galway  
Galway, Ireland  
Enda.Barrett@nuigalway.ie

Enda Howley  
College of Engineering and Informatics  
National University of Ireland, Galway  
Galway, Ireland  
Enda.Howley@nuigalway.ie

Jim Duggan  
College of Engineering and Informatics  
National University of Ireland, Galway  
Galway, Ireland  
Jim.Duggan@nuigalway.ie

ABSTRACT
Solving a finite Markov Decision Process using techniques from dynamic programming such as value or policy iteration require a complete model of the environmental dynamics. The distribution of rewards, transition probabilities, states and actions all need to be fully observable, discrete and complete. For many problem domains a complete model containing a full representation of the environmental dynamics may not be readily available. Bayesian model learning is a technique devised for approximating aspects of the environment in the absence of a complete environmental model. Building on observations through interactions within the environment, an agent can build up an approximation of the required missing model. However this approach can often require extensive experience in order to build up an accurate representation of the true values. To address this issue this paper proposes a method for parallelising a Bayesian model learning technique aimed at reducing the time it takes to approximate the missing model. We demonstrate the technique on learning next state transition probabilities without prior knowledge. The approach is general enough for approximating any probabilistically driven component of the model. The solution involves multiple agents learning in parallel on the same task. Agents share probability density estimates amongst each other in an effort to speed up convergence to the true values.

Categories and Subject Descriptors
H.4 [Artificial Intelligence]: Learning

Keywords
Bayesian Model Learning, Parallel Agent Learning, Reinforcement Learning

1. INTRODUCTION
The solution of a Markov Decision Process (MDP) using methods from Dynamic Programming (DP) requires a complete and fully observable environmental model. When a complete model is available i.e. the set of states, actions, transition probabilities and rewards are known, then the process reduces to a planning problem and can be solved using DP methods. However, often in the real world complete models governing the next state probabilities and distribution of rewards are not known or readily available. In the absence of a complete model two solutions predominate the learning community. The first approach is to attempt to approximate the missing model and solve using traditional DP methods. This is known as model based learning, with many established techniques such as Dyna, Certainty Equivalent Methods and Prioritised Sweeping prevalent in the area. The second approach is to attempt to approximate a control policy directly. This method calculates the value of choosing particular actions in each state prioritising actions which yield the greatest reward in the long run. Commonly known as model free learning, approaches such as Q-learning, SARSA and TD(λ) are widely used.

Typically large reinforcement learning tasks suffer from the curse of dimensionality problem, where the state space grows exponentially with each additional state variable added. Many techniques have been proposed aimed at addressing this problem but particular emphasis has focussed on function approximation methods[21, 15, 3]. Function approximation techniques reduce the problem space by generalising over states and actions not yet visited based on their proximity to ones that have. The agent does not have to directly experience every state in order to learn a policy. These methods greatly enhance the range of domains that reinforcement learning can realistically be deployed, particularly for large real world problems.

The increase in availability of parallel computing resources through cloud and utility computing paradigms has facilitated an alternative approach to solving for large state and actions spaces. Known as parallel learning methods [11], these approaches can potentially offer effective solutions when dealing with large agent learning tasks [5]. To date a number of approaches have been developed for parallelising learning algorithms including Stochastic Gradient Descent [22], SARSA [7] and the dynamic programming techniques, value iteration and policy iteration [5]. Within this domain two main approaches towards the parallelisation of learning techniques have predominated thus far. The first approach involves de-constructing the learning task into subtasks, with each agent learning on a specific part of the problem [14]. The second approach involves agents learning on the same task and sharing their learning experiences with one another [11].

This paper focusses on the second approach where agents learn in parallel on the same task. We propose a parallel
Bayesian model learning approach in which agents share estimates of probability density functions with one another in an effort to speedup the approximation of the true probabilities. In this scenario aspects of the model are missing and through sharing their experiences the agents build up an approximation of the missing parts. However in order to correctly approximate this, large amounts of environmental experience is required. By parallelising the model learning process, the length of time taken to approximate the model is reduced substantially. To facilitate information sharing whilst learning on the specified task, we use the Kullback-Leibler divergence metric to calculate the relative distance between agents’ learned probability estimates and that of a global representation. Individual agents compare their estimates with the global estimate and only share their experience if theirs deviates from the global estimate, above a pre-determined threshold. The approach is highly scalable, given that the amount of information shared amongst the agents reduces over time as a function of the distance between its distribution and that of the global estimate. This paper demonstrates the technique by learning transition probabilities estimates through Bayesian model updates with no prior knowledge. The approach is also general enough to approximate other probabilistic components of the model.

The rest of this paper is structured as follows: Related Research provides an overview of relevant work in this field. Markov Decision Processes and Bayesian Learning details the theory of Markov Decision Processes, and outlines in detail a Bayesian Model Learning algorithm. Learning in Parallel outlines our parallel learning approach to Model learning. Experimental Results empirically examines the performance of the approach, leading to Conclusions & Future Work.

2. RELATED RESEARCH

Previous work by Kretchmar [11] demonstrated the convergence speedups made possible by applying a parallel reinforcement learning approach to action value estimations. He demonstrated this empirically using multi-armed bandit tasks. In this work, the agents share action-value estimates for the different arms while operating on the same bandit task. Each agent has a unique learning experience due to the stochastic nature of the reward distribution. Learning in parallel the agents share information amongst each other and update their knowledge based on their own observations and the global observation. The results clearly demonstrate that the convergence time leading to the discovery of optimal actions is reduced as the number of agents is increased.

Grounds and Kudenko [7] developed an approach for solving a single agent learning task using parallel hardware. The algorithm focuses on sharing value function estimates represented by linear function approximators. Using the Message Passing Interface agents asynchronously transmit messages containing their learned weight values over the previous time period. This paper demonstrated that parallel computing resources applied to a single learning task reduced the time taken to compute good policies over a range of problem domains.

Kushida et. al [12] published a comparative study of three parallel implementation models for reinforcement learning. Two of them utilise Q-learning and one uses fuzzy Q-learning. Their results compared each algorithm's performance against one another and showed an improvement in convergence over the course of the experiments.

Building upon the successes with machine learning and data mining Li and Schuurmans [13] designed a number of parallel learning algorithms which utilise the MapReduce paradigm and execute in parallel. They presented solutions for both classical dynamic programming methods and also for parallelising temporal difference and gradient temporal difference methods.

To the best of our knowledge this is the first attempt aimed at parallelising a Bayesian model learning technique. Previous approaches have focussed on sharing value estimates or linear function approximation representations and combining the results to learn better policies in a shorter time period. One of the benefits of model learning techniques is that, once the missing model has been approximated traditional methods from dynamic programming can be applied to generate solutions. However the greatest challenge with this approach is trying to approximate these missing aspects within a suitable time frame. This paper demonstrates that through learning in parallel approximation times can be greatly reduced enhancing the viability of model based approximation techniques as suitable methods.

3. MARKOV DECISION PROCESSES AND BAYESIAN LEARNING

This section details the Markov Decision Process and the Bayesian learning approach upon which we demonstrate our parallel learning technique.

3.1 Markov Decision Processes

Markov Decision Processes (MDPs) are a particular mathematical framework suited to modelling decision making under uncertainty. A MDP can typically be represented as a four tuple consisting of states, actions, transition probabilities and rewards.

- $S$, represents the environmental state space;
- $A$, represents the total action space;
- $p(.)$, defines a probability distribution governing state transitions $s_{t+1} \sim p(.|s_t, a_t)$;
- $q(.)$, defines a probability distribution governing the rewards received $R(s_t, a_t) \sim q(.|s_t, a_t)$;
- $S$ the set of all possible states represents the agent’s observable world. For many problems the agent learning experience is broken up into discrete time periods. At the end of each time period $t$ the agent occupies state $s_t$. The execution of the chosen action, results in a state transition to $s_{t+1}$ and an immediate numerical reward $R(s_t, a_t)$. The state transition probability $p(s_{t+1}|s_t, a_t)$ governs the likelihood that the agent will transition to state $s_{t+1}$ as a result of choosing $a_t$ in $s_t$. The numerical reward received upon arrival at the next state is governed by $q(s_{t+1}|s_t, a_t)$ and is indicative as to the benefit of choosing $a_t$ whilst in $s_t$.

The solution of a MDP results in the output of a policy $\pi$, denoting a mapping from states to actions, guiding the agent’s decisions over the entire learning period. Algorithm 1 outlines a value iteration algorithm for determining optimal policies for a finite MDP. The algorithm involves the
iterative computation of the value of each state \( V(s) \). Once the true value of each state has been approximated the optimal policy can be determined by choosing actions greedily with respect to the approximated state values [20].

As stated previously in the specific case where a complete environmental model is known, i.e. \( \{S, A, p, q\} \) are fully observable, the problem reduces to a planning problem [16] and can be solved using traditional dynamic programming techniques such as value iteration. However if there is no complete model available, then one must either attempt to approximate the missing model (Model Based Reinforcement Learning) or directly estimate the value function or policy (Model Free Reinforcement Learning). Model based techniques involve the usage of statistical techniques in order to approximate the missing model [8]. In this paper we adopt a model based learning approach known as Bayesian model learning which attempts to approximate the missing model.

### Algorithm 1 Value Iteration

Initialize \( V \) arbitrarily

repeat

\[ \Delta \leftarrow 0 \]

For each \( s \in S \)

\[ v \leftarrow V(s) \]

\[ V(s) \leftarrow \max_a \sum_{s'} P_{s,a}^n [R_{s,a}^n + \gamma V(s')] \]

\[ \Delta \leftarrow \max(\Delta, |v - V(s)|) \]

until \( \Delta < \Theta \)

Output policy \( \pi \)

For all \( s \in S \)

\[ \pi(s) = \arg\min_a \sum_{s'} P_{s,a}^n [R_{s,a}^n + \gamma V(s')] \]

### 3.2 Bayesian Model Learning

If one does not possess a complete environmental model, then one approach to solving a MDP is to approximate the missing model through actual experience of operating within that environment. In a MDP different aspects of the model may be hidden from the learning agent, these can range from a partial or hidden state space, to transition probabilities or the distribution of rewards. Partially Observable Markov Decision Processes (POMDPs) are a particular framework for modelling environments where one only has a partial or limited view of the state space. In this environment the agent’s ability to directly observe the underlying state is restricted. The agent maintains a probability distribution over the set of possible states, based on actual observations and observation probability estimates. In addition to the state space limitations, transition probabilities and the distribution of rewards may also be unknown. For demonstrative purposes we deal strictly with the case where the states, actions and rewards are fully observable, finite and stationary, but the transition probabilities are unknown and must be learned. The goal is therefore to approximate the transition probabilities through experience and solve via the value iteration algorithm presented in Algorithm 1. Depending on the problem scale and underlying environmental dynamics varying amounts of experience will be required in order to gain a measure of “ground truth”. In order to reduce the amount of time taken to approximate the probabilities we propose deploying multiple agents in parallel which share probability estimates over time. Bayesian learning is a statistical learning method which provides a mathematical formula to determine how one should change one’s existing beliefs in light of new evidence. For a given random variable \( X \) the posterior probability \( P'(X = x|e) \), is the probability that \( X \) is equal to a value \( x \) given experience \( e \). In the general case the posterior probability is computed using Bayes theorem

\[
P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}
\]

which requires a conditional probability \( P(X|Y) \) and two unconditional probabilities \( (P(Y), P(X)) \) to compute a single conditional posterior probability \( P(Y|X) \). Our goal is to compute the next state probabilities for a given action \( a \) in state \( s \). This translates to \( P'(s = s'|a, s) \) the posterior probability that \( s \) is equal to the next state \( s' \) given action \( a \). The Bayesian learning approach presented here is similar to what was proposed by Spiegelhalter et. al. [19] and further extended by Doshi et.al [6]. It uses a modified Bayes rule, where all that is required to compute the posterior probability is an initial prior probability and subsequent environmental experience. The approach involves maintaining an experience counter \( \text{Expc} \) for each state variable. Each time the agent chooses an action \( a \) within state \( s \) and observes the next state \( s' \), the experience associated with the transition is incremented by 1. Knowledge is accumulated iteratively over time for each action selected within a state. Through observing the responses the agent builds up a model of the underlying dynamics. Equations 2 and 3 define the update rules for approximating the transition probabilities from experience.\(^1\)

\[
P'(s = s'|a, s = s) = \frac{P(s = s'|a, s = s) \times \text{Expc} + 1}{\text{Expc}'}
\]

Equation 3 ensures that probability distribution over the total number of states sums up to 1.

\[
P'(s = s'|a, s = s) = \frac{P(s = s'|a, s = s) \times \text{Expc}}{\text{Expc}'}
\]

To illustrate the workings of this approach we consider the simple 4 × 4 gridworld from Sutton and Barto [20] shown in Figure 1. In this world there exists a total of 14 possible states. Within each state the agent may choose from a set of four possible actions (Up, Down, Left, Right). Each action causes a deterministic state transition in the direction of that action, i.e. \( P_{1,2}^{\text{right}} = 1 \) and \( P_{1,14}^{\text{right}} = 0 \). All actions which take the agent off the grid leave its state unchanged. All state transitions result in a reward of −1 except those \( \text{Expc}' = \text{Expc} + 1 \)

\[\text{Expc}' = \text{Expc} + 1\]

\[\text{Expc}' = \text{Expc} + 1\]

\[\text{Expc}' = \text{Expc} + 1\]

\[\text{Expc}' = \text{Expc} + 1\]

\[\text{Expc}' = \text{Expc} + 1\]
that result in a terminal state. In true Bayesian form it is
assumed that the transition probability \( P'_{s' \mid a, s} \) for all states
and actions is unknown. All state transitions initially are
given an equal probability of occurrence \( P'_{s' \mid a, s} = \frac{1}{|S|} \). This
results in an initial equi-probable prior distribution for all
states and actions i.e. \( P'_{s' \mid a, s} = 0.04 \) and \( P'_{14 \mid a, s} = 0.04 \).
Through experience the agent adjusts its estimates of the
transition probabilities. A truncated example is given below,
showing the changing probability estimates over time as the
agent updates its experience of choosing action \( \text{right} \) in state 1.

The experience counter is initialised to 1.

\[
t = 0
\]

\[
P'(s' = 2|a = \text{right}, s = 1) = \frac{0.04 \times 1 + 1}{2} = 0.52
\]

\[
P'(s' = 3|a = \text{right}, s = 1) = \frac{0.04 \times 1}{2} = 0.02
\]

\[
\vdots
\]

\[
P'(s' = 14|a = \text{right}, s = 1) = \frac{0.04 \times 1}{2} = 0.02
\]

\[
t = 1
\]

\[
P'(s' = 2|a = \text{right}, s = 1) = \frac{0.52 \times 2 + 1}{3} = 0.68
\]

\[
P'(s' = 3|a = \text{right}, s = 1) = \frac{0.02 \times 2}{3} = 0.0133
\]

\[
\vdots
\]

\[
P'(s' = 14|a = \text{right}, s = 1) = \frac{0.02 \times 2}{3} = 0.0133
\]

The updated posterior probability \( P' \) from the previous
episode serves as the new prior probability for the next learn-
ing episode as shown in \( t = 1 \). As the agent gains greater
experience the learned transition probabilities converge to
the true probabilities. Further information regarding this
method can be found here [6].

4. LEARNING IN PARALLEL

Our approach to learning in parallel, involves agents sharing
estimates of probability densities over time in order to
speedup convergence to the true transition probabilities. The
agents attempt to approximate the transition probabilities independently of one another. The approach only works
when simulated offline or when guarantees preventing the
agents from affecting each others experience are upheld. In
other words, the transitions between states can not be ef-
fected or interfered by the decisions of other agents learning
in parallel.

Employing a Bayesian model learning algorithm, each agent
maintains a local posterior probability estimate \( P' \) and has
access to a global estimate \( G \) representing the collective
knowledge of all other learning agents. This allows the
agents to separate their own personal experience, from that
of all others. In order to facilitate learning in parallel the
agents need some measure of the relative difference between
their estimate and the global estimate. The Kullback-Leibler
(KL) divergence, sometimes referred to as information gain
or relative entropy is a technique capable of measuring the
distance between two probability distributions. Given two
probability distributions \( P \) and \( Q \), the KL divergence is
given by

\[
D_{KL}(P \parallel Q) = \sum_i \ln \frac{P(i)}{Q(i)}
\]

Note that the KL divergence is not a true metric and is not
symmetrical, meaning that the KL divergence from \( P \) to \( Q \)
is not equal to the KL divergence from \( Q \) to \( P \). If \( P \) and \( Q \)
are identical then \( D_{KL} = 0 \).

Algorithm 2 depicts the steps involved in our parallel
Bayesian model learning approach. Firstly the prior proba-
bility \( P \) is defined as equi-probable across all possible next
states. The global estimate \( G \) is also initialised as equi-
probable. \( P \) forms the initial prior for the Bayesian update
rules defined in Equations 2 and 3. The communications
arrays \( \text{comms}_{in} \) and \( \text{comms}_{out} \) are initially set to \( \emptyset \). \( M \)
represents the overall MDP albeit with the vector of transi-
tion probabilities \( p \) initially equal to the prior probability
\( P \). Updating this model through learning is the objective of
the algorithm. The action chosen within each state is gov-
erned by the policy \( \pi \) which is calculated using the Value
Iteration algorithm specified in Algorithm 1. The optimal
policy \( \pi^* \) mapping each state to the corresponding optimal
action can only be computed if the transition probabilities
are known. Thus the solution of the MDP via Value Itera-
tion is dependent upon the accuracy of the model learning
process. For each time step, the agent occupies a state \( s \)

Algorithm 2 Parallel Model Learning

Initialise \( P'_{s' \mid a, s} \), \( G_{s, a, s} = \frac{1}{2} \)
Initialise MDP \( M \) with \( p \leftarrow P \)
repeat

\[
\pi \leftarrow \text{ValueIteration}(M)
\]

Choose \( a \) from \( s \) using policy \( \pi \)
Take action \( a \), observe \( r, s' \)
Update \( P' \) according to Eq(2) and Eq(3)
\( P \leftarrow P' \) (posterior becomes new prior for next iteration)

Include global knowledge and combine experience
\( P' = \frac{P' \times P_{\text{Expc}} \times G_{s, a, s}}{P'_{s' \mid a, s} + P_{\text{Expc}} \times G_{s, a, s}} \)

\( M \leftarrow M \) with \( p \) replaced by \( P' \)
if \( \| \text{D}_{kl}(P', G) \| > \theta \) then
Add \( P' \) to \( \text{comms}_{out} \)
end if

Transmit \( \text{comms}_{out} \) to all agents
Receive \( \text{comms}_{in} \) from other agents
if \( \text{comms}_{in} \neq \emptyset \) then
for all \( Q \in \text{comms}_{in} \) do
Compute global estimate
\( G = \frac{Q \times P_{\text{Expc}} \times G_{s, a, s}}{P'_{s' \mid a, s} + P_{\text{Expc}} \times G_{s, a, s}} \)
end for
end if

until \( \| \text{D}_{kl}(P', G) \| > \theta \)

and chooses an action \( a \) according to policy \( \pi \). The agent
receives a numerical reward \( r \) upon execution of the action
and transitions to the next state \( s' \) with an unknown prob-
ability. The agent, observing the next state, computes the posterior probability $P'(s = s'|a, s)$ using Eq.(2) in light of the new experience. Next the agent modifies the posterior by combining the experience of all the other learning agents and the agent’s own experience. The KL divergence between $P'$ and $G$ gives the agent a measure of the distance between the two distributions, global and local. If this distance is greater than a predetermined threshold then it adds $P'$ to the outgoing communications array $comm_{out}$. This information is then transmitted to all other learning agents to incorporate within their own estimates.

All information received from other agents is accessed via $comm_{in}$ and during each learning step the agent integrates this information into the global estimate value. The final step involves replacing the model transition probabilities $p$ with the latest learned estimates $P'$. This facilitates the computation of improved policies over the next iteration of the algorithm. Thus as the agent gains experience and improves its approximation of the transition function, the policy converges towards the optimal.

The principle benefit of this approach is that the agent does not have to directly visit each state in order to approximate the model as the agents share their experiences.

5. EXPERIMENTAL RESULTS

To demonstrate the effectiveness of the approach at improving the convergence time for approximations of the transition probabilities, we experimentally validate our approach using a stochastic Gridworld example [20] and a virtual resource allocation problem supporting both cost and quality constraints. In the first problem the transition probabilities belong to a discrete non-uniform distribution. The second set of experiments simulates a real world problem, where real trace data sets are used to ground the simulations in a realistic setting. Resources are allocated in response to changes in both user requests and virtual machine performance.

5.1 Stochastic Gridworld

For the first experiment we evaluate performance on a stochastic Gridworld problem. Similar to the Gridworld example depicted in Figure 1, the cells of the grid represent the different states of the environment. In each state, the agent can choose from a set of four possible actions (Up, Down, Left, Right). Each chosen action has a probability of 0.6 of moving the agent one step in that direction and a probability of 0.2 of moving the agent at right angles to the chosen direction. The reward on each transition is 0. An action that takes the agent off the grid leaves its state unchanged but incurs a negative reward of −1. A number of special states are dispersed randomly throughout the grid. This number depends on the overall size of the gridworld. Once the agent enters a special state, it is deterministically transported to a corresponding state somewhere else on the grid and receives a large numerical reward.

Figure 2 plots the Kullback-Leibler divergence between the combined (global) weighted transition probability distribution of all the agents against the true transition probabilities over 200 episodes. With this experiment the number of agents is increased from 1 to a maximum of 10. Figure 2 clearly shows a performance improvement as the number of agents increases from 1 to 10. This experiment validates the performance speedups achievable for learning transition probabilities in this domain.

5.2 Virtual Machine Allocation

The second experiment simulates learning approximate transition probability estimates in a real world setting. The problem selected focusses on a cloud resource allocation problem, where resources are allocated in response to user requests and performance variabilities. Multiple agents are deployed in parallel to decrease the time taken to approximate the next state transition probabilities. The experiment uses real http response time data sets and an open loop queueing user request model to simulate a cloud resource allocation problem.

5.2.1 Datasets and Resource variability

Cloud based virtual services exhibiting performance variabilities due to virtual machine interference has been well documented in the literature [9, 17]. The sharing of resources amongst the respective VMs is handled by the Virtual Machine Monitor (VMM) [4], an independent domain level monitor which has direct access to the underlying hardware. Xen is a popular open source virtualisation framework, supporting a wide range of guest operating systems and is used by a large number of cloud providers including Amazon Web Services. Xen facilitates a software layer known as the Xen Hypervisor which is inserted between the server’s hardware and the operating system. This allows the physical host to deploy multiple VMs in isolation, decoupling the operating system from the physical host. However whilst virtualisation technologies such as Xen provide excellent security, fault and environmental isolation they do not ensure performance isolation. Koh et. al.[10] state that there are three principle causes of this interference. The first cause is due to the fact that each independent VM on the hypervisor has its own resource scheduler which is attempting to manage shared resources without the visibility of others. Secondly guest operating systems and applications inside a VM have no knowledge about ongoing work in co-located VMs and are unable to adapt in response. Thirdly some hypervisors such as Xen, offload operations including I/O operations to service VMs. This particularly affects I/O-intensive applications as the Xen hypervisor forces all I/O operations to pass through a special device driver domain forcing the context to switch into and out of the device driver domain causing interference.

Cedexis [1] the multi-cloud deployment strategists gather
large quantities of http response time data on a daily basis, measuring cloud performance from disparate geographical locations across multiple different cloud providers. Customers of Cedexis agree to the placement of a small piece of JavaScript on a busy page of their web application. Once a user visits the particular page a script automatically downloads with the page and is executed client side within the browser\(^2\). A list of URLs containing the location of different files on different cloud platforms is returned and the user randomly chooses from amongst them. The response time between the user and cloud is recorded. This technique offers a more holistic view of response time performance from the user’s perspective as both the cloud performance and the local ISP performance is considered. A Bitcurrent\(^3\) report in 2011 using data from Cedexis over a seven day period, provide insights into the variability associated with disparate user http response time across different regions of the globe. This report was compiled using almost 300 million data points, ranging over nine different globally distributed cloud providers. For the simulated experiments below we base our virtual machine performance on the global response time distribution table published in this work [2].

5.2.2 MDP Formalism

To facilitate agent learning for a cloud resource allocation problem we define a simple state action space formalism. The state space \(S\) consists of a single state variables \(S = \{q\}\).

- \(q\) is the average http response time for the service over the period under consideration. This is indicative as to the quality of the service and is defined in the SLA.

The agent’s action set \(A\) contains the set of all possible actions within the current state. The agent can choose to add, remove or maintain the amount of virtual machines allocated to support the service. Rewards are determined based on a defined Service Level Agreement (SLA) which is directly related to performance. The overall reward allocated per time step is given by the following equations.

\[
C(a) = C_r \times V_a + \left\{ \sum_{i=1}^{\nu} (C_r \times V_i) \right\}
\]

\[
H(a) = P_r \times \left\{ \begin{array}{ll}
1 + \frac{p' - sla}{sla} & \text{if } p' > SLA \\
0 & \text{else}
\end{array} \right.
\]

\[
R(s', a) = C(a) + H(a)
\]

\(C_r\) is the cost of the resource, this is variable depending on the type, specific configuration and region. \(V_i\) represents an individual virtual machine instance, with \(V_s\) representing the specific virtual machine allocated, deallocated or maintained as a result of action \(a\). \(H\) is the overall penalty applied as a result of violating the specified SLA. \(P_r \in \mathbb{R}\) represents a defined penalty constant incurred through violation of the SLA. The SLA comprises a guaranteed level of service response time agreed between the service provider and consumer. The total reward \(R(s', a)\) for choosing action \(a\), resulting in \(s'\) is the combination of the cost of execution and any associated penalties.

\(^2\)The procedure only executes when the browser is in an idle state

\(^3\)Is now CloudOps research

![Figure 3: Kullback-Liebler divergence for single state and single action over 3 separate runs](image1)

![Figure 4: Kullback-Liebler divergence from true probabilities for different agent numbers](image2)
number increasing from 2 to 10 agents. As the number of agents increases the time taken to estimate the true probabilities reduces significantly.

6. CONCLUSIONS & FUTURE WORK

This paper presents an approach which successfully parallels a Bayesian model learning algorithm. The approach is based on agents sharing information with one another in an effort to learn approximate probability distributions in true Bayesian form. This paper has shown empirically using a stochastic gridworld example and a cloud resource allocation problem, that convergence speedups can be achieved through the described parallel learning approach. Although this paper focusses on learning approximate transition probability distributions, the approach is general enough to approximate any probabilistically driven component of the model. For future work we wish to investigate the approach’s performance with non-stationary probabilities. We wish to determine its efficacy and ability to "keep up" with the changing dynamics of the underlying probabilities. Finally we also wish to further develop techniques to reduce the amount of information shared between agents learning in parallel.

7. ACKNOWLEDGMENTS

The authors would like to gratefully acknowledge the continued support of Science Foundation Ireland. The authors also wish to thank Cedexis for the use of the performance data for the purposes of this research.

8. REFERENCES