Multi-model Approach to Non-stationary Reinforcement Learning

Samuel P. M. Choi, Dit Yan Yeung, Nevin L. Zhang
Department of Computer Science
Hong Kong University of Science and Technology
Clear Water Bay, Kowloon
Hong Kong
{pmchoi,dyyeung,lzhang}@cs.ust.hk

ABSTRACT
This paper proposes a novel alogrithm for a class of non-stationary reinforcement learning problems in which the environmental changes are rare and finite. Through discarding corrupted models and combining similar ones, the proposed algorithm maintains a collection of frequently encountered environment models and enables an effective adaptation when a similar environment recurs. The algorithm has empirically compared with the finite window approach, a widely-used method for non-stationary RL problems. Results have shown that our algorithm consistently outperforms the finite window approach in various empirical setups.

KEY WORDS
Reinforcement Learning, Non-stationary Environment

1 Introduction

Learning to perform sequential decision tasks in a complex environment is non-trivial, especially when the environment is not known in advance. Reinforcement learning (RL) [4, 9] is a computational approach to such a task through learning from interaction. Thus far, most existing RL researches are focussed on stationary Markovian environments; i.e., the underlying dynamics of the environment depend solely on the current state and are independent of time. Non-stationary environments, on the contrary, refer to the stochastic environments in which the underlying parameters may vary over time.

Non-stationary problems are very common in the real world. Consider a robot rover which explores in an unvisited planet. When roaming around, the rover may encounter various types of weathers and terrains (e.g. uphill, downhill, and craters). In order to navigate in a desired manner, the rover may need different sequence of control actions for each environment. Hence, it would be ineffective to treat all these environments as a whole, as what is typically done in traditional RL. A more plausible way would be to build a separate model for each environment, so that the learned model and the computed policy can be deployed when a similar environment recurs. We therefore propose a multi-model RL algorithm for such non-stationary environments. We maintain a collection of environment models by discarding corrupted models and combining incomplete ones. To be precise, our proposed work can be considered as a sub-class of non-stationary environments of which changes are rare and limited to a finite set of Markov decision processes (MDPs).

Markov decision processes (MDP) is the fundamentals of RL and have been commonly used for characterizing stochastic environments under which optimal decisions are to be made. Formally, an MDP is defined as a 4-tuple (S, A, T, R), where S represents the set of states, A the set of actions, T the transition function, and R the reward (or cost) function. Typically, the transition function T is denoted as $T(s, a, s')$, which represents the transition probability from state s to state s' by taking action a. The reward function is denoted as $R(s, a)$, which gives the reward, usually between 0 and 1, for any state-action pair. If all the parameters of an MDP remain unchanged over time, then the MDP is said to be *stationary*. To solve a *Markov decision problem*, one is given an MDP and some performance criterion to represent the long-term accumulated reward. A policy that optimizes the performance criterion is computed as a solution to the problem.

Most existing algorithms for non-stationary RL (e.g. [3, 1]) are specific to certain sub-class of non-stationary environments. The most commonly used approach to various types of non-stationary RL problems is the finite window approach [4] (FWRL). The key idea of the finite window approach is to maintain the N most recent experience instances for building the internal model so as to track the environmental changes. The learning system is therefore able to adapt faster to the environment and to reach an acceptable, but possibly suboptimal, policy. Due to its simplicity and reasonable performance on various types of non-stationary environments, the finite window approach is a good bench mark for comparing non-stationary RL algorithms.

2 Multi-model Reinforcement Learning

The main idea of the multi-model reinforcement learning (MMRL) is as follows. For every N time steps, MMRL constructs an environment model and maintains a number of previously learned models in the model library. At each time step, MMRL collects the current experience and computes a posterior probability distribution over the environment models it maintains. The choice of action is then determined according to the probability distribution and the exploration rate.

In other words, the MMRL algorithm contains four main components: model construction, model maintenance, model identification, and decision making. Whereas the third and the fourth components are invoked at every step, the first and the second components are executed only every N steps. In the following, these four components are examined in detail.

2.1 Model Construction

Building environment models is relatively easy as existing model-based RL techniques (e.g. prioritized sweeping [7]) can be directly applied. In our implementation, we keep the most recent N tuples of experience by using a fixed-width history window and then build the environment models by the certainty equivalent method [5]. When a model is constructed, the optimal policy (or a near-optimal policy, depending on the allowed computation time) is also computed simultaneously.

The main difficulty in model construction is that the window width N must be carefully chosen. If the constant N is too small, there will be insufficient data for constructing an accurate environment model. On the contrary, if N is too large, the collected experiences will possibly come from more than one environment. As a result, the learned model cannot truly represent the underlying environment. Worse still, there is often no appropriate number that works for all cases. Two mechanisms that attempts to reduce the sensitivity of the chosen N are therefore devised.

The first mechanism is used to discard incorrect environment models; or equivalently, to abandon discrepant data collecting from multiple distinct environments due to a large window width. An *integrity test* is devised for determining the purity of the data. The second mechanism aims to combine inaccurate environment models (due to insufficient data from a small window width) so as to form a more accurate model. A *cross-integrity test* is used to decide whether two environment models should be merged. Both the integrity and cross-integrity tests are based on the standard statistic methods; namely, the Chi-square and the Kolmogorov-Smirnov tests [8].

Using the Chi-square and the Kolmogorov-Smirnov tests, the integrity test attempts to measure the confidence on the obtained data indeed coming from one environment mode. The returned value (*integrity*) is subsequently used to decide whether the induced environment model should be stored into the model library. If the integrity is higher than a pre-determined threshold, the learned model will be saved; or if otherwise, discarded.

Figure 1 details the integrity test. An integrity test first divides a given observation sequence $\mathcal{O}$ into two equal halves. Then for every state s and action a, the state transition counts of the two data sets are compared by using the Chi-square test. The function CHI-SQ$(\cdot)$ in Figure 1 returns the significance level of the difference of two data sets. This function is well-defined if both data sets are not empty. Similarly, the significance levels of the obtained rewards are computed by the Kolmogorov-Smirnov test KS-TEST$(\cdot)$. The integrity value of the observation sequence is then computed by the average of these probabilities.

```
INTEGRITY($\mathcal{O}$: the observation sequence): real
$count \leftarrow sumprob \leftarrow 0$
divide $\mathcal{O}$ into 2 equal halves $\mathcal{O}_1$ and $\mathcal{O}_2$.
**for** $s \in \mathcal{S}$ **do**
    **for** $a \in \mathcal{A}$ **do**
        **for** $s' \in \mathcal{S}$ **do**
            $c^1_{s,a}(s') \leftarrow$ the number of $(s, a, s')$ tuple in $\mathcal{O}_1$.
            $c^2_{s,a}(s') \leftarrow$ the number of $(s, a, s')$ tuple in $\mathcal{O}_2$.
        **end for**
        $p \leftarrow$ CHI-SQ$(c^1_{s,a}, c^2_{s,a})$
        **if** $p$ is well-defined,
            $sumprob \leftarrow sumprob + p$
            $count \leftarrow count + 1$
        **endif**
        $r^1_{s,a} \leftarrow$ the obtained rewards from $s$ and $a$ in $\mathcal{O}_1$.
        $r^2_{s,a} \leftarrow$ the obtained rewards from $s$ and $a$ in $\mathcal{O}_2$.
        $p \leftarrow$ KS-TEST$(r^1_{s,a}, r^2_{s,a})$
        **if** $p$ is well-defined,
            $sumprob \leftarrow sumprob + p$
            $count \leftarrow count + 1$
        **endif**
    **end for**
**end for**
**return** $sumprob/count$
```

Figure 1. Computing the integrity of the data sequence

Nevertheless, the integrity test cannot always discriminate inconsistent data. In particular, the test will fail if the two data partitions are composed of equal portions of stationary environments. (an example is shown in Figure 2). While such a coincidence is rare, it could still happen in practice. Fortunately, this problem can be partially addressed by another mechanism — the *model combination*.

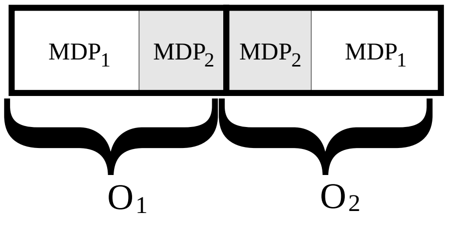

Figure 2. An example showing that the integrity test fails.

2.2 Model Maintenance

MMRL maintains a collection of environment models in the model library in order to provide a quick adaptation to the previously engaged environments. The number of environment models stored in the model library grows and shrinks dynamically. In this section, we discuss how two environment models representing the same environment can be combined into a more accurate one.

For every N steps, a new environment model is constructed and examined by the integrity test. If the model passes the test, it will conduct a cross-integrity test with every model stored in the model library. If the highest value exceeds a pre-defined threshold, the corresponding pair of models will be combined. This *model combination* mechanism identifies and combines similar models so as to increase the accuracy of the models stored in the library.

There is another situation that also requires the learned models being combined and discarded; i.e., when the model library is full. In that case all the environment models with integrity greater than a threshold are identified, and a cross-integrity test is conducted on every pair of the selected models. If the highest cross-integrity value exceeds a certain value, the corresponding pair of models are combined. Otherwise, the environment model with the lowest integrity is discarded. This scheme provides MMRL with an opportunity to reorganize the environment models stored in the model library, so that more accurate models can be formed.

We now examine the details of the cross-integrity test. A cross-integrity test signifies the confidence on the difference between two environment models. The cross-integrity test is computed by the Chi-square test; however, it differs from the integrity test in two aspects. The first difference is that the cross-integrity test considers only state transitions, whereas rewards are ignored. It is because reward sequences are not stored in the model library due to the storage consideration. The second difference is that the confidence value of the cross-integrity is not computed by the average probability. Instead, the probabilities are weighted by the number of the state transitions. This additional weighting is needed because environment models are often comprised of unequal amount of data. Figure 3 depicts the complete cross-integrity routine.

Inheriting from the limitation of the Chi-square and Kolmogorov-Smirnov tests, the cross-integrity test might also fail to discriminate two environment models that differs only marginally. In other words, it is possible that two slightly different models are being mistakenly combined. Despite such a limitation, the cross-integrity test in practice works reasonably well in identifying similar environment models. It is because the probability of the alternation in optimal policy due to a slight change in model parameters is usually very small.

```
CROSSINTEGRITY($\mathcal{M}_1, \mathcal{M}_2$: environment models): real
$avgprob \leftarrow 0$
for $s \in \mathcal{S}$ do
    for $a \in \mathcal{A}$ do
        for $s' \in \mathcal{S}$ do
            $c^1_{s,a}(s') \leftarrow$ the number of $(s, a, s')$ tuple in $\mathcal{M}_1$.
            $c^2_{s,a}(s') \leftarrow$ the number of $(s, a, s')$ tuple in $\mathcal{M}_2$.
        end for
        $prob \leftarrow$ CHI-SQ($c^1_{s,a}, c^2_{s,a}$)
        if $prob$ is well-defined,
            $avgprob \leftarrow avgprob + \dfrac{p \cdot \sum_{s'} c^1_{s,a}(s') \cdot \sum_{s'} c^2_{s,a}(s')}{\sum_s \sum_a (\sum_{s'} c^1_{s,a}(s') \cdot \sum_{s'} c^2_{s,a}(s'))}$
        endif
    end for
end for
return $avgprob$
```

Figure 3. Computing the cross integrity of two models

2.3 Model Identification

Model identification is for determining in which environment model the learning agent currently resides. This can be achieved by maintaining a probability vector, each entry of which describes the posterior probability of an environment model stored in the model library. By observing the state transitions and the rewards received, the probability vector can be updated by the Bayes rule. That is,

$$P(M_i|D) = \frac{P(D|M_i)\, P(M_i)}{P(D)}$$

where D is the most recent experience tuple $(s_{t-1}, a_{t-1}, s_t, r_t)$, and M_i denotes an environment model stored in the model library. Since we are interested only in the relative probability distribution over the models M_i, the prior probability of data, $P(D)$, can be ignored through normalization. The prior probability, $P(M_i)$, can be estimated by the number of data stored in M_i. The likelihood $P(D|M_i)$ can also be computed easily according to the M_i's transition and reward functions. For every instance of experience, the probability vector $P(M_i|D)$ is then updated by multiplying $P(M_i|D)$, the transition probability, and reward probability. The updated vector values are then normalized to ensure that $\sum_i P(M_i|D) = 1$. In other words, the updating rule for the probability vector is:

$$P_t(M_i|D) = k \cdot P(r_t|s_{t-1}, a_{t-1}) \cdot T(s_{t-1}, a_{t-1}, s_t) \cdot P_{t-1}(M_i|D) \qquad (1)$$

where k is the normalization factor.

3 Decision Making

Once the probability vector is updated, the best action is the one which maximizes the expected utility; namely,

$$action = \arg\max_{a \in A} \sum_i P(M_i|D) \cdot Q_i(s,a) \qquad (2)$$

where A is the action space, M_i is the i-th environment model stored in the model library, and $Q_i(s,a)$ is the Q-value of M_i given state s and action a. This decision rule is equivalent to the one used by Chrisman and McCallum [2, 6]. While this simple decision rule provides only sub-optimal solutions due to ignoring the future uncertainty, it requires substantially less computation and performs reasonably well in most cases.

Unlike Chrisman and McCallum's formulation, our decision rule also consider the current model estimated in the history window, but only after sufficient experiences are accumulated. In addition, an exploration scheme is employed according to the maximum uncertainty on the probability vector. More specifically, there is a probability of one minus maximum certainty to randomly choose an action which is rarely taken at the current state. In addition, if the highest certainty points to the estimated model, more explorations should take place. The pseudo code of the MMRL algorithm is depicted in Figure 6.

MMRL requires four threshold settings. Like many heuristic methods, choosing appropriate thresholds is an art and often requires fine-tuning for different tasks. Fortunately, according to our experiments, the sensitivity of these thresholds (c.f. Figure 6) were quite low. For ϵ_1 and ϵ_2, a low value and a high value are typically used respectively. A reasonable choice is any value smaller than 0.2 for ϵ_1, and any value greater than 0.8 for ϵ_2. For ϵ_3 and ϵ_4, a wider range of values can be adopted. Generally speaking, a high threshold value guides MMRL to proceed cautiously, and as a result, more accurate models are learned. On the contrary, a low threshold value allows MMRL to adapt to the environment faster, but there is a risk that the learned models are only crude approximations of the reality. From our experiments, a reasonable value for both ϵ_3 and ϵ_4 was between 0.5 and 0.75.

4 Empirical Results

In this section, the effectiveness of MMRL is studied empirically. Experiments have been conducted on a number of randomly generated environments of various sizes and complexities, and the performance of MMRL is compared with the finite-window RL approach (FWRL). FWRL is chosen for baseline comparison since it is conceptually simple and yet performs reasonably well on various types of nonstationary environments. In the following, two typical results are shown. Findings on the empirical results are then summarized and discussed.

4.1 Experiment One: A Trivial Case

The experiment commenced with a simple hidden-mode nonstationary environment where all the environment models occurred first and were properly fitted into the history window. This setting was to simulate the situation that the environment models were given in advance, either by human experts or by pre-training. In addition, no maintenance was done in the model library, and thus the main purpose of the experiment was to study the effectiveness of the model identification and the decision making components. This experiment contained 3 randomly generated MDPs, each of which was composed of 5 states and 4 actions, and the discount factor γ was 0.95. For both algorithms, the window size were set to 1000. No limit was being imposed on the size of the model library. The exploration rate for FWRL was set to a fixed value of 0.3.

Figure 4 shows the empirical result. The environmental changes were depicted below the x-axis, and the window boundaries were shown as the grid lines. Each plot indicated the average utility over all states, given the learned policy at a particular time instant. An approximated average of the optimal value function was also included for reference. In this experiment, environment models were learned quite accurately, and the performance of MMRL was far better than the FWRL. In the first 3000 time steps, MMRL was in the model construction stage, and FWRL performed comparably to MMRL. From the 3000 time step onward, MMRL quickly adapted to the environmental changes, while FWRL performed rather unsatisfactorily.

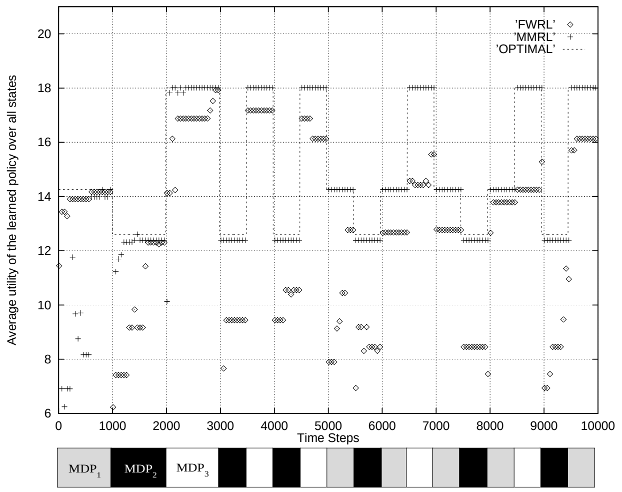

Figure 4. Empirical result 1

4.2 Experiment Two: A General Case

The second experiment was conducted on a larger problem (3 MDPs, 20 states, 10 actions), and the environments were no longer properly aligned with the history window. Window size for both algorithms was set to 3000. The capacity of the model library was set to 5. The threshold ϵ_1 equals 0.1 while ϵ_2 equals 0.9. Both ϵ_3 and ϵ_4 were set to 0.5.

Figure 5 illustrates the experimental result. In the first 15000 time steps, the first 5 environment models were built and stored, with integrity values 0.5215, 0.2676, 0.5386, 0.4962, and 0.5172 respectively. Up to this point, no environment models were discarded nor combined. At time 18000, the model library was full, and MMRL had to determine which environment model was to be discarded or to be combined with another one. After examining all the models in the library, MMRL failed to find a pair of models which satisfied the model combination criteria. Therefore, the model constructed from time 3000 to 5999, which had the lowest integrity (0.2676), was discarded. At time 21000, a new environment model was created, which had an integrity value of 0.5003 and a cross-integrity value of 0.5737 with the first model in the model library. MMRL combined the two environment models to replace the old one. Note that thereafter, the performance of the learning agent improved for all MDP_1 environments since a more accurate model was formed. At time 24000, an environment model with integrity 0.3554 was learned, and MMRL decided to discard it because it had the lowest integrity among all the learned models. At time 27000, a new model gave an integrity value 0.5154 and a cross-integrity value 0.6031 were combined with the third learned environment model. At time 30000, MMRL discarded the current learned model (integrity = 0.3833), and at time 33000, MMRL combined the current model with the first environment model. As a result, the first environment model contained 9000 data points. Since then, no better environment model was formed and the model library had no further update. The empirical result showed that although the set of learned environment models were not yet completely accurate, MMRL still performed consistently better than FWRL in terms of the adaptation rate and the obtained rewards.

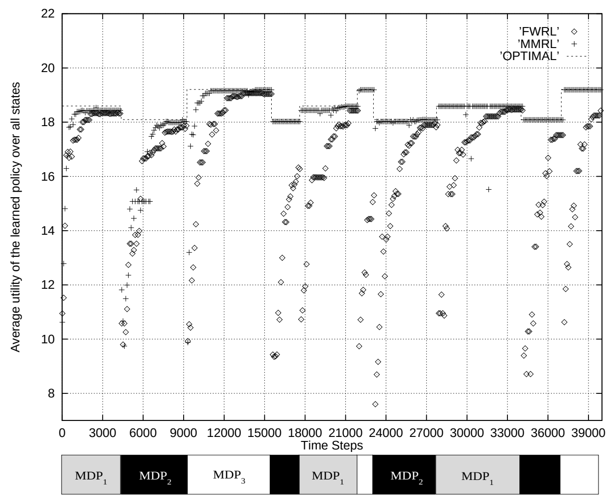

Figure 5. Empirical result 2

5 Additional Issues

In this section, a number of additional issues that affect the performance of MMRL are identified and discussed. Some future extensions to the algorithm are also proposed.

Window Size: It is probably the most important question to MMRL. As the performance of MMRL relies on an accurate environment model, the window size being chosen has a crucial effect on its performance. As mentioned earlier, while a large window is capable of learning a more accurate environment, it is also prone to cover more than one environment. A small window, on the contrary, has a higher chance of learning a pure environment model, but it inevitably reduces the model accuracy. Although the model combination scheme alleviates such a problem, a very small window with a low exploration rate will still cause an inaccurate estimation and missing entries in the environment model, which in turn cause problems in computing the probability vector as well as the optimal policy. Therefore, in order to acquire a reliable estimate of the environment, the window size should be proportional to the product of the sizes of the state and action spaces. In the future, we will attempt to find the boundary of the environment models by using a variable-length window.

Exploration vs. Exploitation: The trade-off between exploration and exploitation is a notorious issue even in stationary environments. In nonstationary environments, this issue becomes more crucial because more explorations must take place in order to notice the environmental changes. In the current implementation, a simple exploration scheme on the basis of environmental uncertainty is used. It is possible to employ a more sophisticated exploration scheme, such as assigning an exploration rate proportional to the fluctuations of the probability vector values on the learned environment models. The rationale behind that is when confidence changes frequently among the learned models, it is likely that a new environment is engaged and thus more explorations are needed.

Maintaining the Model Library: Empirical results show that the library size has a relatively small effect on the performance of the algorithm, so long as its capacity is reasonably large for holding the frequently occurring environments. However, when the model library is full, one must consider either combining two similar environment models stored in the library or simply discarding one. The current implementation uses heuristic measures, the integrity tests, to determine whether an environment model is to be combined or discarded. It is possible to devise a more robust scheme by incorporating other MDP metrics, such as cross entropy, and the policy difference. In addition, other techniques used in memory caching such as the least-recently referred, or the least-referred, may also be employed.

6 Conclusion

A new algorithm, MMRL, is proposed for non-stationary RL problems. MMRL assumes infrequent changes in a finite number of environments, and dynamically maintains the frequently encountered environment models over time. It is akin to Chrisman's perceptual distinctions approach (PDA) [2] in several aspects; in particular, the use of the fixed-width window, the use of statistical techniques for testing two distributions, maintaining a probability vector similar to the belief state, and taking uncertainty into account for deciding the next course of action. An important difference between the two is that MMRL combines and discards environment models while PDA only splits its states. Moreover, MMRL has its unique features such as the model library, the integrity tests, and a model maintenance scheme. Preliminary experiment results have suggested the superiority of the MMRL over the finite-window approach in terms of both the adaptation rate and the obtained rewards.

References

[1] S. Choi, N. Zhang, and D. Yeung. Reinforcement learning in nonstationary environments. In R. Sun and L. Giles, editors, *Sequence Learning: Paradigms, Algorithms, and Applications*, 264–287. Springer-Verlag, 2001.

[2] L. Chrisman. Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. In *AAAI-92*, 1992.

[3] P. Dayan and T. Sejnowski. Exploration bonuses and dual control. *Machine Learning*, 25(1):5–22, October 1996.

[4] L. Kaelbling, M. Littman, and A. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, May 1996.

[5] P. Kumar and P. Varaiya. *Stochastic Systems: Estimation, Identification, and Adaptive Control*. Prentice-Hall, 1986.

[6] A. McCallum. Overcoming incomplete perception with utile distinction memory. In *Tenth International Machine Learning Conference*, Amherst, MA, 1993.

[7] A. Moore and C. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less real time. *Michine Learning*, 13, 1993.

[8] W. Press, B. Flannery, S. Teukolsky, and W. Vetterling. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, 1988.

[9] R. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, 1998.

```
1. Initialize the model library.

2. Clear the history window.
   While the history window is not full,
      • update the probability vector (Equation 1)
      • determine and carry out an action according to the
        probability vector and the exploration rate (Equation 2)
      • receive the next state $s_{t+1}$ and immediate reward $r_{t+1}$.
      • update the current environment model $M'$.
      • perform one or several steps of model-based RL on $M'$
   end while

3. If the data integrity $< \epsilon_1$,
      the model $M'$ is discarded.
   else
      compute the cross-integrity of $M'$ and $M_i$, for all $i$.
      If the highest cross-integrity $> \epsilon_2$,
         combine $M'$ with the corresponding $M_i$.
      else
         If the model library is not full,
            save the current environment model.
         else
            for all saved models with integrity $> \epsilon_3$,
               • compute the cross integrity for every pair of models,
                 and keep the pair with the highest cross-integrity.
            end for
            If the highest cross-integrity $> \epsilon_4$,
               combine the pair of the models.
            else
               discard the model with the lowest integrity.
            endif
         end if
      end if
   end if

4. Reassign the probability vector according to the models stored
   in the model library.

5. Goto step 2.
```

Figure 6. Multi-model reinforcement learning algorithm