Hierarchical Reinforcement Learning with Context Detection (HRL-CD)

Yiğit E. Yücesoy and M. Borahan Tümer

Abstract—**A reinforcement learning (RL) agent mostly assumes environments are stationary which is not feasible on most real world problems. Most RL approaches adapt slow changes by forgetting the previous dynamics of the environment. Reinforcement learning-context detection (RL-CD) is a technique that helps determine changes of the environment's nature which the agent with the capability to learn different dynamics of the non-stationary environment. In this study we propose an autonomous agent that learns a dynamic environment by taking advantage of hierarchical reinforcement learning (HRL) and present how the hierarchical structure can be integrated into RL-CD to speed up the convergence of a policy.**

Index Terms—**Reinforcement learning, autonomous agent, hierarchical reinforcement learning, non-stationary environment, betweenness centrality, prioritized sweeping.**

I. Introduction

Reinforcement learning (RL) is a behavioral learning method in which an agent interacts with an environment by choosing actions and this environment returns a reward signal and a new state. The aim of the agent is to maximize its cumulative reward by attaining some goal state. The environment is generally represented by a Markov Decision Process (MDP). The agent-environment interaction proceeds until the agent reaches a terminal state or a pre-specified number of actions are taken [1]. The goal state is a terminal state that the agent is expected to attain through the execution of a series of actions. The agent is said to learn this series of actions if it always selects (in a greedy policy) to execute this series of actions. If the agent accrues a maximum of cumulative reward by executing this specific series of actions then it is said to learn the optimal policy.

In problems with a large or continuous state space, hierarchical approaches are advantageous because the agent can consider not just one task, but a whole range of sub-tasks, solve them independently, and combine their individual solutions to solve the whole task. Also the agent is capable of executing primitive actions as well as higher-level, temporally-extended actions, called options. Options are represented in terms of Semi-Markov Decision Processes (SMDPs) and establish a close analogy with actions and MDPs they are represented with. SMDPs provide a flexibility for the options in terms of the variable execution timing possibilities which also gives way to using actions concurrent with options as single time-step options [2]-[4].

Manuscript received December 22, 2014; revised April 21, 2015. This work is supported by the Marmara University Research Fund, BAPKO, D-Type project.

Yiğit E. Yücesoy is with the Halic University, Istanbul, Turkey (e-mail: efe@ycsoy.com).

M. Borahan Tümer is with the Marmara University, Istanbul, Turkey (e-mail: borahan.tumer@marmara.edu.tr).

Decomposing a problem into smaller tasks greatly improves RL methods; on the other hand, it raises another problem: how are sub-goals of options to be extracted? Sub-goals can be seen as bottlenecks or doorways in an environment [4]. If the agent can discover these bottleneck regions and learn policies to reach them from within a set of initial states (i.e., the initiation set) during the initial stages of learning, it can use these policies for more effective exploration as well as refine quicker its overall policy. McGovern and Barto used diverse density [5], Şimşek and Barto employed the concept of betweenness centrality from the graph theory [6], and Menache *et al*. used graph based Min-Cut algorithm to find sub-goals in an environment [7].

In a non-stationary environment an effective RL algorithm must be able to keep up with the changes to find an optimal policy, which turns out to be a problem for classic RL algorithms. Choi *et al.* developed a method called Multiple-Model Reinforcement Learning (MMRL) [8] which assumes a fixed number of dynamics and known by the designer. This assumption is not practical and unpretentious for real world problems. In order to overcome this problem, Silva *et al*. developed Reinforcement Learning with Context Detection (RL-CD) algorithm for solving RL problems in a non-stationary environment by detecting environmental changes and creating new or selecting existing partial models for the current state of the environment [9].

Hierarchical reinforcement learning is used for large scale problems via creating a new hierarchy by separating the main problems into sub-problems. Also RL-CD provides the agent to perceive the differences of the environment's dynamics. This study extends RL-CD by constructing over a hierarchical structure to improve convergence speed to a fit policy.

II. Related Work

A. Reinforcement Learning

Reinforcement Learning (RL) is a behavioral learning method where the learner, the agent, does not have any knowledge about its current state or consequences of actions. The agent learns how to behave from the immediate –negative or positive- reward received as a response to each action [10]. RL can be modeled by Markov Decision Processes (MDP), a sequential, discrete time, decision making framework which can be specified by four variables (s, a, s', r) where; $s \in S$ is the current state of agent, $a \in A_s$ is the selected action in action set of state s, $s' \in S$ is the next state which is responded by environment and r is the immediate reward of transition from state s to s' with action a.

With this setup, the problem can be defined in sequential discrete time steps, t=0, 1, 2..., T. In every step, the agent observes the current state s_t, then performs an action at from a and receives a new state s_{t+1} and an immediate reward r_{t+1}.

At each step the agent selects an action among possible actions by taking into account the probabilities from policy, π. The agent applies the policy to determine which action a to choose on state s. In other words, the policy $\pi(s, a)$ denotes the probability of taking action a in state s under policy π [9]. The agent's goal is to find the best policy to maximize the total reward.

B. Dynamic Programming Value Iteration

In the context of MDP, *dynamic programming* (DP) offers algorithms to evaluate a policy provided that a perfect model of the environment is available. DP can be used to compute value functions and an optimal policy can be gained from optimal value function, V^*. $P_{ss'}^{a}$ is the transition probability of reaching state s' from state s when taking action a and $R_{ss'}^{a}$ reward of transition from state s to state s' via action a [9].

$$V(s) = \max_a \sum_{s'} P_{ss'}^{a} \left[R_{ss'}^{a} + \gamma V(s') \right] \tag{1}$$

This operation is called *backup* because a state-value is updated by using approximate future or successor state-values. Applying one backup to every state is called a *sweep* which is iterated to converge to an optimal value function V^* [11].

C. Prioritized Sweeping

Prioritized Sweeping (PS) is a *memory-based* RL algorithm which creates a model of the environment during the learning phase. In general, the model is used by the agent to predict how the environment will respond to its actions, this is why the model contains next state — s' and reward signal — r for a state-action -(s,a) tuple. With this information, PS effectively updates the value table of (s,a) pairs where only those pairs' values get adjusted which move the environment to an (s,a) pair with a non-zero value, which is the main idea behind this algorithm. A priority queue is kept to update (s,a) pairs in a sorted fashion by the amount of their value adjustment. Θ is an application-dependent threshold parameter to determine if an (s,a) pair picks up a value adjustment big enough to deserve to be in the priority queue and actually get updated. (If the (s,a) pair is already in the queue, the newly calculated priority gets compared with the old priority and the maximum one is updated for that (s,a) pair. After updating the value of the (s,a) pair selected from the queue by its priority, priorities of predecessors to the selected state are calculated and if eligible enough they are pushed into the queue. In this way updates are propagated backwards until there is no priority left significant enough to handle [12].

D. Hierarchical Reinforcement Learning

HRL is a RL technique devised to cope up with the *curse of dimensionality* caused by problems with a large or continuous state space by decomposing the learning process into *hierarchies* using divide-and-conquer approach. Either manually by the designer or by some algorithm prior to the learning [4], or, as we present, autonomously by the agent as a part of the learning [10], the environment is divided into a number of regions that facilitate the split of the main task into a number of sub-tasks with well-defined sub-goals.

In a lower level hierarchy, all sub-goals are learned within the corresponding regions using "flat" RL techniques, and the policy π_{ij} that leads in region i to sub-goal sg_j is stored — along with a set of possible start states describing the relevant region of application and a termination condition specifying where and with what probability the policy ends — as a complex action or the term we use in this work, an option.

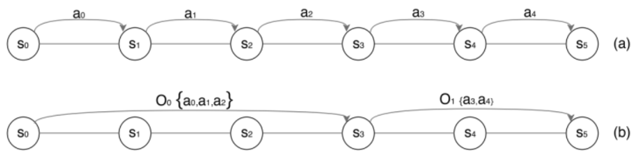

Fig. 1. Time steps of classical MDP (a) and MDP with options (b).

In an upper level hierarchy, equipped with a set of options executable in different regions of the environment, the agent is capable of moving with "larger steps" (i.e., in a temporally extended fashion as shown in Fig. 1(b)) any time it selects one of these options in relevant states.

In order to define an option three components must be included: $I \subseteq S$, initiation set which is a portion of the environment, π: $S \times A \rightarrow [0,1]$ is the defined policy for the option's behavior and β:$S^+ \rightarrow [0,1]$ is the termination condition of the option. An option o can be used in all states which are defined in the initiation set [2]. When an option terminates the agent selects another option. Note that primitive actions are considered as special case of options which consist of one single time step [13].

E. Betweenness Centrality

On a social network, importance of a person raises with how many people are connected through that person. Making an analogy to the graph theory, some nodes are more critical than others since they lie on the shortest paths between many node pairs. In this sense they form a bottleneck playing a central role for other nodes' communication. In other words, a point is considered to be central if it falls between other points' shortest paths [14], [15].

$$g(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}} \tag{2}$$

where s, t and v are some nodes on a grid, σ_{st} is the number of shortest paths from s to t and $\sigma_{st}(v)$ is the number of shortest paths from s to t which pass on node v. $g(v)$ is the betweenness centrality of the node v. In order to calculate betweenness centrality of a node, all the shortest paths between each node pair node are needed to be calculated except adjacent nodes.

F. Reinforcement Learning with Context Detection

On a classical RL approach, changes can be adapted by an agent via forgetting previously learned policy and learning new dynamics of an environment which works only if the nature of an environment changes slowly and does not come up repetitively. However, when the use of some historical experience addressing a previous state of the environment is required, the agent has to be able to acquire the corresponding knowledge about the environment. Otherwise if the environment changes and any of the previous experience do not correspond to the new state, the agent must detect the need of a new model as mentioned in RL-CD. To apply the RL-CD method, dynamism of an environment should be divided into stationary dynamics. However the changes of the environment do not have to be known by the agent. The purpose of RL-CD is to perceive the differences automatically and divide the environment into stationary parts [16]. Therefore RL-CD requires four conditions to be satisfied; 1) dynamics of an environment should be separated into several environments which have their own nature, 2) the differences which are caused by the environment's nature cannot be observable by the agent but they can be predicted, 3) the changes which occur in the environment are independent from the actions of the agent and 4) the changes do not occur frequently.

RL-CD creates models, *m*, for every different characteristic of the environment. *Quality* of a model, *E*, is a value that shows the extent to which the estimate of this previously acquired model matches with the current environment. The agent uses the model with the *highest quality*, E_{max}. When there is no model that has a higher *E* than a pre-specified minimum match quality threshold, i.e., E_{min} is less than *the minimum quality threshold* that means a new model must be created by the agent [8].

III. Hierarchical Reinforcement Learning with Context Detection

RL-CD detects environmental changes and as a reaction, creates a new model of the environment or assigns a previously learned model for the agent. Once a new model is created, it is considered as a new problem, in other words with every new model, a new problem emerges. In this study, HRL is applied to speed up the convergence to a possibly optimum policy of every distinct partial model. This enhancement in speed is acquired by naturally isolating clusters of options to be used only at specific models.

In order to autonomously divide the main problem into smaller tasks, sub-goals should be discovered by the agent without the interference of a designer. We use betweenness centrality, which requires a perfect model; hence, the agent should acquire a rather close model to a perfect model to discover sub-goals. In other words if the agent has an ability to perceive the environment as a whole sub-goals can be discovered using betweenness centrality.

Betweenness centrality can detect important nodes of a graph already generated by any model based algorithm, and since RL-CD also needs a model of the environment, a model based algorithm, PS, is used as the learning algorithm in this study. Fig. 2 shows a model created by the agent from experience which is modeled as a graph to apply betweenness centrality.

The agent must assess the accuracy of the model which must reflect the environment as accurately as possible to gather better sub-goal states. At the end of each episode, betweenness centrality is iterated by the agent to find candidate sub-goal states until the states discovered to be potential sub-goals by the agent are stably identified.

In order to estimate the accuracy of discovered sub-goals we use variance of sub-goal counts ($Var[C_{SG}]$) on every episode which decreases with the increasing model accuracy. In other words, the agent's model of the environment must be accurate enough to create options or sub-tasks.

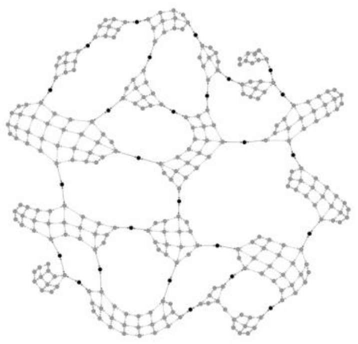
Fig. 2. Model of an environment constructed by the agent.

When the search for sub-goals is finished, the agent creates initiation sets for options by using the model of the environment. After an initiation set and the corresponding sub-goal are defined for an option, the agent can use a model-based algorithm, e.g., DP Value Iteration (DVI) to learn the defined portion of the environment. DVI requires and works on a perfect model of the environment provided by the initiation set of the option so an option can be learned apart from the main problem.

Once the options of a partial model are learned by the agent, learning continues over the hierarchical structure exploiting the options learned so far. Thereby for the partial model on every state, an option can be selected among primitive actions. On that account, on the partial models which represents the environment with sufficient accuracy the agent learns using options. On the other hand other partial models that do not satisfy the mentioned criteria have to continue learning over primitive actions, i.e., one time-step options.

Hence in order to construct a setup for HRL-CD apart from the classical RL-CD, options, current ε values for ε-greedy selection, and step counts for every episode must be stored and isolated in the corresponding partial model.

On the other hand, on a dynamic environment, termination condition for options (*β*) is not sufficient because the agent must make a new decision when the environment changes other than continuing the option selected for the previous nature of the environment. For this reason, when the agent detects a change in the environment after choosing a proper model, the option executed by the agent is terminated to make a new decision for the current state of the environment.

IV. Empirical Results

A. Setup for the Experiments

Limitations of the problem selection for this work are mostly dictated by the RL-CD algorithm's restrictions about the environment [8]. We conduct our experiments on a grid world problem to demonstrate that 1) the algorithm works, 2) it outperforms RL-CD. Further the grid world is used since it is suitable enough for visualization purposes so that environmental changes and reactions of the agent can be efficiently observable. Dynamism is incorporated to the problem by reversing or rotating the directions of the actions as indicated in Fig. 3.

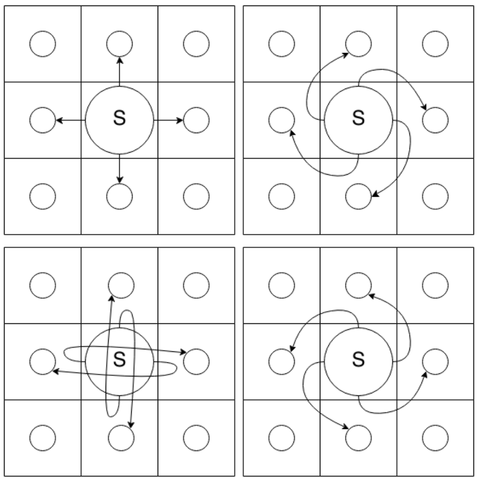

Fig. 3. Different dynamics of the grid world problem; default, right rotated, reversed, left rotated.

Prioritized sweeping parameters for updating Q function are chosen as learning rate, α=0.2 and discount factor, γ=0.95 and threshold, θ=0.1.

Policy for choosing actions is constructed on ε-greedy strategy. The value of ε is started at 0.7 and decreased to 0.01 as the learning continues.

$$\varepsilon(t) = 0.01 + 0.7e^{\left(-\frac{t}{100}\right)} \tag{3}$$

Exponential Decay function (3) is implemented for determining ε value for each time-step which decreases ε value to 0.01 over time.

For RL-CD, $E_{\min}$ and ρ values are set to -0.4 and 0.1 respectively.

TABLE I: ENVIRONMENT STATE SPACES OF EXPERIMENTS

State Space	Environment Dynamics
100	2
225	2
400	2
400	4
625	2
900	2
900	4
1225	2
1600	2
2025	2
2500	2
3025	2
3600	2

In this study, we conducted 1300 experiments with various state-spaces shown in Table I. With each state space size, 100 experiments are performed where each 100-experiment set consists of 50 runs of PS with RL-CD and 50 runs for HRL-CD.

B. Results

In initial stages, HRL-CD operates essentially on primitive actions to learn and solve the problem until the agent gathers enough knowledge about the environment. That is why, some parts of the environment (usually those parts closer to terminal states) are experienced by primitive actions until options are created and learned by the agent. Accordingly, this fact results, for small environments with small state space, in too fast a convergence to a policy before options can be learned. This is not any different than RL-CD, but mechanisms for hierarchical structure become redundant for such a problem. Thus when dealing with small environments HRL-CD does not have any advantages over classical RL-CD.

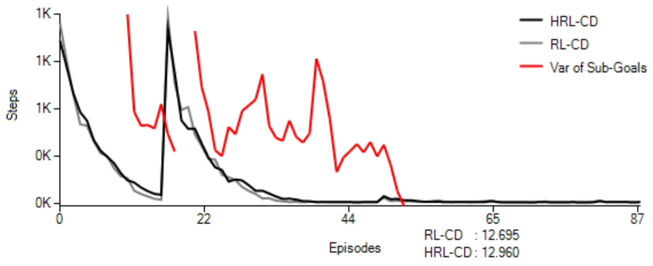

Fig. 4. An environment with 100 states.

As a typical example to the above remark we conduct an experiment with a relatively small (i.e., 10×10) state space. HRL-CD suffers from the problem mentioned above. In Fig. 4 we show that HRL-CD does not offer an advantage over RL-CD for this problem. At this point, the hierarchical structure becomes redundant for environments of approximately 100 states and below for the given problem.

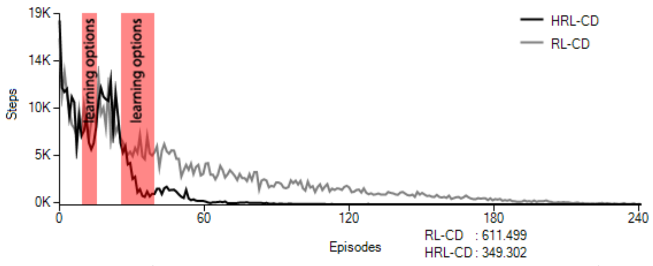

Fig. 5. Range of learning options in 50 trials with an environment of 900 states.

Fig. 5 shows the results of the experiment with 30×30 states which better presents the influence of the hierarchical structure over RL-CD. Until options are built HRL-CD and RL-CD show the same tendency. HRL-CD agent outperforms RL-CD after starting to utilize options and the convergence to the learned policy significantly accelerates. Learning of options occurs at the interval between the two red vertical lines shown in Fig. 5.

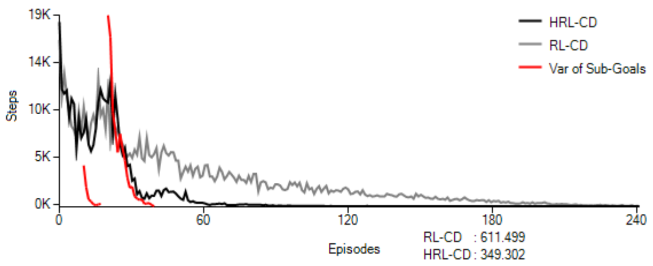

Fig. 6. Learning curve of the agent on an environment with 900 states.

The effect of the options on learning can also be interpreted by step counts of episodes as stated in Fig. 5 or can be observable by the variance of step counts through the learning process of the problem as shown on Fig. 6. As we take into account the sub-goal counts to determine if the model is accurate enough, the fall of the variance of sub-goal counts ($Var[C_{SG}]$) under a specific threshold triggers the construction of the options.

On Fig. 6 $Var[C_{SG}]$ starts no sooner than 10th episode because we average over the last 10 episodes to calculate $Var[C_{SG}]$. The accuracy of the environment model, measured by the $Var[C_{SG}]$ per episode, is continuously checked to detect the episode at which the model represents the environment closely enough. This specific episode is where the construction of options may begin since at this very point, the model sufficiently closely represents the real environment. Since, after learning options, we do not need to assess the accuracy of the model any more, the calculation of $Var[C_{SG}]$ stops and the learning process of the main problem continues also making use of the available options.

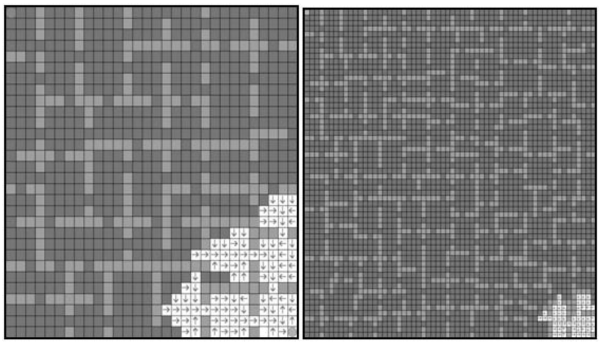
Fig. 7. Captures of 30×30 and 60×60 environments on the time options are learned. Dark areas indicate the corresponding portion of the environment that does not learned by the agent.

Fig. 7 shows two environments with 900 and 3600 states, respectively. White areas at the lower right corner are learned by the HRL-CD agent using primitive actions and while learning the remaining part (i.e., the dark areas) of the environment the options are also employed. The larger environment (the one at the right) can gain more performance from options because a larger area can be learned with the involvement of hierarchical structures.

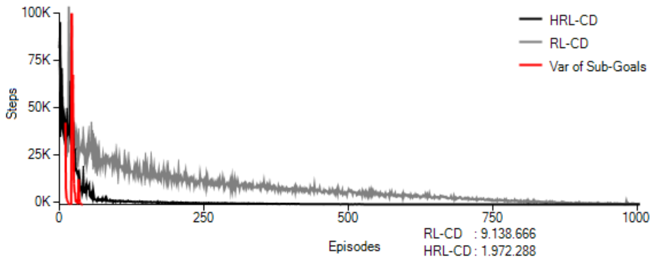

Fig. 8. Learning curve of the agent on an environment with 3600 states.

In Fig. 8, the HRL-CD agent arrives at the point at which $Var[C_{SG}]$ falls under the preset "model maturity" threshold and allows for the construction of options in average before episode 29 (12%) and episode 30 (3%) for environments with 900 and 3600 states as shown in Fig. 6 and Fig. 8, in respective order.

Instead of comparing states, we can compare updated action-values which can provide a more accurate evaluation. Table II tabulates the number of Q values learned i.e., updated to a value different than zero. This measurement gives a more accurate rate about the learned portion of the environment. Thus a better comparison can be achievable.

For instance, we already know that HRL-CD does not offer an improvement when considering an environment with 100 states. Table II shows that approximately 44% of the problem is experienced by the agent using only primitive actions and this 44% of experience can even include the solution of the main problem. But as the problem become larger, for example the environment with 3600 states, construction of the options occurs after the agent goes through approximately 1.2% of the main problem, which leaves 98.8% of the problem to be solved by options.

TABLE II: EFFECT OF ENVIRONMENT SIZE ON USAGE OF OPTION/ PRIMITIVE ACTION RATE

State Space	Learned Q Values	Missed Q Values	Learned by	
			Primitive Actions	Options
100	172	217	44.20%	55.80%
225	200	684	22.60%	77.40%
900	182	3404	5.10%	94.90%
3600	166	14213	1.20%	98.80%

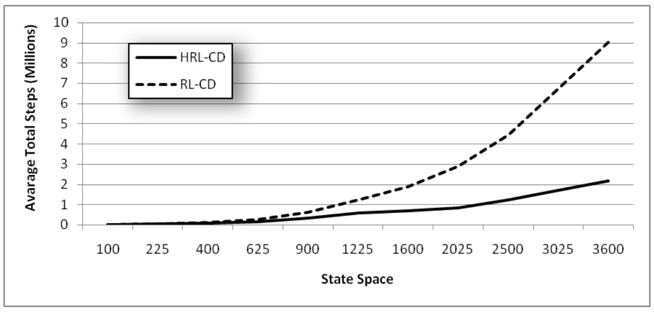

Fig. 9. Total average steps with various environment sizes.

Results of various sizes of experiments are presented in Fig. 9. We show in this figure that as the state space increases, effectiveness of HRL-CD over classical RL-CD grows. The results shown on Fig. 9 can be considered as alternative evidence supporting our claim that the sooner the options are started to be used, the quicker the convergence into a satisfactory policy occurs. The sooner the agent has access to options during learning, the faster HRL-CD agent converges than RL-CD agent provided they operate on the same environment.

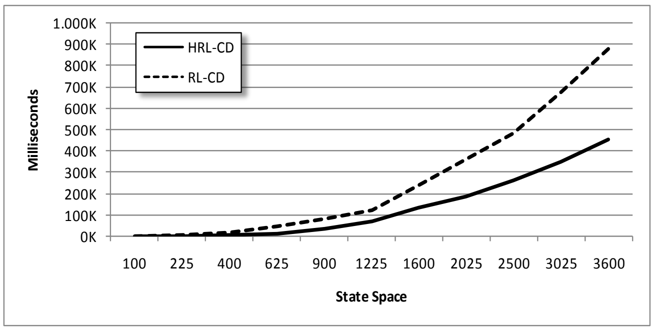

Fig. 10. Average time in milliseconds with various environment sizes.

We have also compared the actual execution times of both algorithms. We have employed the *StopWatch* class of .NET Framework to assess the time as a measurement and run the program under Windows 7 operating system. We illustrate the time curves on Fig. 10. One should note here that both the implementation of the algorithm and the concurrency properties the relevant operating system provides extremely affect to the time consumed by the execution of options and primitive actions. Even the results presented on Fig. 10 show that HRL-CD outperforms RL-CD, total time steps, shown in Fig. 9, is a more accurate measure than the actual execution times of both algorithms.

V. Conclusion

In this work we focused on deterministic, discrete, dynamic environments, which have different dynamics that occurs infrequently and independent from the agent. We implemented an autonomous agent that can learn a dynamic environment based on RL-CD technique without intervention of a human. We also take advantage of hierarchical reinforcement learning for constructing the algorithm which has been used in the literature to speed up the learning process [2]. In order to keep the agent autonomous, hierarchical structure is maintained by betweenness centrality with the idea proposed at [5].

By exploiting RL-CD and HRL with betweenness centrality we implemented an algorithm called *Hierarchical Reinforcement Learning with Context Detection* which shows significant improvement especially for environments with large state-space. Even though we did not observe any progress on environments with small state-space, as the size of the environments grows, HRL-CD outperforms RL-CD with increasing significance. This behavior of the agent is the result of the preparing phase for the hierarchical structure. In order for the HRL-CD agent to autonomously create a hierarchy, the agent must gather some information about the environment, but on small problems, the RL-CD agent solves the problem before HRL-CD agent constructs a hierarchy which results in the same or a poorer performance for the HRL-CD agent.

Our implementation of HRL-CD separates and stores options in their corresponding model and prevents the usage of options among models which is justifiable and a safe approach since every model addresses a different dynamic of the environment. But if we consider a chance that different models can include the same or similar initiation sets, learning these initiation sets separately becomes redundant. Soni et al. offers a solution to transfer policies from one domain to another by adopting homomorphism [17]. As a future work, the approach in [17] can be applied to HRL-CD to transfer policies of the options among models for the different dynamics of the environment.

On the other hand, restrictions about the environment stated by the RL-CD do not define a strict rule about the differences of the dynamism of the environment. A similar study may be run where the amount of the difference of various stationary portions and the time spans of individual dynamisms of the dynamic environment are two other system variables.

References

[1] A. Mcgovern, R. S. Sutton, S. Singh, D. Precup, and B. Ravindran, "Hierarchical optimal control of MDPs," in *Proc. the Tenth Yale Workshop on Adaptive and Learning Systems*, 1998, pp. 186-191.

[2] R. S. Sutton, D. Precup, and S. Singh. (1999). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, [Online]. pp. 181-211. Available: http://dx.doi.org/10.1016/S0004-3702(99)00052-1

[3] A. McGovern and A. G. Barto, "Automatic discovery of subgoals in reinforcement learning using diverse density," in *Proc. the 18th International Conference on Machine Learning*, 2001, pp. 361-368.

[4] Ö. Şimşek, "Behavioral building blocks for autonomous agents-description identification and learning," Ph.D. Thesis, University of Massachusetts, Amherst, 2008.

[5] O. Şimşek and A. G. Barto, "Betweenness centrality as a basis for forming skills," Technical Report TR-2007-26, University of Massachusetts, Department of Computer Science, Amherst, MA, 2007.

[6] I. Menache, S. Mannor, and N. Shimkin, "Q-cut — dynamic discovery of sub-goals in reinforcement learning," *Machine Learning: ECML*, pp. 187-195, 2002.

[7] S. P. M. Choi, D. Y. Yeung, and N. L. Zhang, "Multi-model approach to non-stationary reinforcement learning," *Artificial Intelligence and Soft Computing*, ACTA Press, pp. 357-160, 2001.

[8] B. C. Silva, E. W. Basso, A. L. C. Bazzan, and P. M. Engel, "Dealing with non-stationary environments using context detection," in *Proc. the 23rd International Conference on Machine Learning ICML*, 2006, p. 2.

[9] R. S. Sutton and A. G. Barto, "Reinforcement learning: An introduction," Cambridge, MA: MIT press, 1998.

[10] K. L. Pack, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," arXiv preprint cs/9605103, 1996.

[11] B. G. Andrew and S. Mahadevan, "Recent advances in hierarchical reinforcement learning," *Discrete Event Dynamic Systems*, vol. 13, no. 4, pp. 341-379, 2003.

[12] M. W. Andrew and C. G. Atkeson, "Prioritized sweeping: Reinforcement learning with less data and less time," *Machine Learning*, vol. 13, no. 1, pp. 103-130, 1993.

[13] S. Martin and D. Precup, "Learning options in reinforcement learning," *Abstraction, Reformulation, and Approximation*, Springer Berlin Heidelberg, pp. 212-223, 2002.

[14] B. Ulrik, "A faster algorithm for betweenness centrality," *Journal of Mathematical Sociology*, vol. 25, no. 2, pp. 163-177, 2001.

[15] N. E. J. Mark, "A measure of betweenness centrality based on random walks," *Social Networks*, vol. 27, no. 1, pp. 39-54, 2005.

[16] D. de Oliveira, A. L. Bazzan, B. C. da Silva, E. W. Basso, L. Nunes, R. Rossetti, and L. Lamb, "Reinforcement learning based control of traffic lights in non-stationary environments: A case study in a microscopic simulator," *EUMAS*, December 2006.

[17] S. Vishal and S. Singh, "Using homomorphisms to transfer options across continuous reinforcement learning domains," *AAAI*, vol. 6, 2006.

M. Borahan Tümer received his B.S. and M.S. degrees both in computer engineering from Boğaziçi University (İstanbul, Turkey) in 1987, and from Istanbul Technical University, İstanbul, Turkey in 1990, respectively and his Ph.D. degree in electrical and computer engineering from Marquette University, Milwaukee, WI in 1998. Dr. Tümer is an associate professor of the Computer Engineering Department at Marmara University. Dr. Tümer's current research interests are in learning systems with an emphasis on reinforcement learning and sequential decision making, learning automata, adaptive medical signal processing, neural networks.

Yiğit Efe Yücesoy is currently an M.Sc. student at Marmara University and he is working as a research assistant at Halic University, Turkey. He received his B.Sc. degree in computer engineering from the Department of Engineering, Halic University, Istanbul in 2009.