2014 International Joint Conference on Neural Networks (IJCNN)
July 6-11, 2014, Beijing, China

Sharing Information on Extended Reachability Goals Over Propositionally Constrained Multi-Agent State Spaces

Anderson V. de Araujo and Carlos H. C. Ribeiro
Divisao de Ciencia da Computacao
Instituto Tecnologico de Aeronautica - ITA
Sao Jose dos Campos, SP - Brazil
Email: {andva, carlos}@ita.br

***Abstract*—By exchanging propositional information, agents can implicitly reduce large domain state spaces, a feature that is particularly attractive for Reinforcement Learning approaches. This paper proposes a learning technique that combines a Reinforcement Learning algorithm and a planner for propositionally constrained state spaces, that autonomously help agents to implicitly reduce the state space towards possible plans that lead to a goal whilst avoiding irrelevant or inadequate states. State space constraints are communicated among the agents using a common constraint set based on extended reachability goals. A performance evaluation against standard Reinforcement Learning techniques showed that by extending autonomous learning with propositional constraints updated along the learning process can produce faster convergence to optimal policies due to early state space reduction caused by shared information on state space constraints.**

***Index Terms*—Multi-Agent, Cooperative Agents, Reinforcement Learning, Q-Learning, Planning, Markov Decision Processes, Extended Reachability Goals.**

I. Introduction

In recent years, concepts such as decentralization and autonomy have been concentrating much of the attention of researchers in different areas of Computer Science. The pros of decentralized and autonomous software are already well known and have been demonstrated in numerous applications. As far as decision making is concerned, there are acknowledged advantages in considering environments with individuals that cooperate with each other, trying to achieve a single objective or independent goals. Panait and Luke [1] showed an extensive study of learning systems with cooperative agents, presenting examples focused on Evolutionary Computation, Robotics, Reinforcement Learning, etc.

In the context of autonomous learning, this paper considers an extended form of the standard Markov Decision Process (MDP) [2], [3], [4] with propositional constraints on the state space that are communicated among agents as the learning goes on.

In a standard MDP, a state transition model is assumed to be known beforehand, but this is not always the case in real world applications [5]. Reinforcement Learning (RL) is a common research area which addresses this issue, focusing on the agent interaction with the environment as a mechanism for gathering information about the domain structure. Q-Learning [6] is a typical RL algorithm that inherits a notorious difficulty of RL algorithms in general to deal with large state spaces, due to a need for balancing the control objective and the model-free estimation of the domain structure based solely on exploration. Here, we propose a technique which incorporates propositional constraints on the state space, using Q-learning-based algorithms and employing information exchange between agents. By doing so, the objective is to reduce the overall exploratory need, thus improving the performance of the learning algorithm. To constrain the state space, extended reachability goals (ERG) [7] are used. ERGs are comprised by two expressions: one to be preserved during the iteration, and another that describes a goal state. Both are composed by propositions that describe the environment states.

The rest of this paper is organized as follows: Section II introduces the definitions and concepts that were used to develop the proposed solution. Section III details the definition of the adopted model and the applied planning strategies. The developed approach, the algorithms and techniques are also in this section.The experimental set up is defined in Section IV, and the results and a comparative analysis between different algorithms are in Section V. Finally, Section VI summarizes the key features and the main contributions of the proposed approach.

II. Background

A. Markov Decision Process and Reinforcement Learning

A Markov Decision Process (MDP) [2], [3], [4] is a formal model for synchronous interaction between an agent and its environment. At every step the agent observes the current state of the environment and decides to execute one action. The execution of the selected action takes the agent to a new state of the environment and produces

978-1-4799-1484-5/14/$31.00 ©2014 IEEE

a reinforcement (a merit value associated to the state-action pair). The interaction between the agent and the environment continues until a stopping criteria is met.

MDPs are primarily used to model sequential decision making in stochastic environments, and are generally applied in planning and optimization in environments involving uncertainties, such as problems in Robotics, Economics, etc.

A model $\mathcal{M}$ for a standard MDP can be defined as: $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$, where:

- $\mathcal{S} \neq \emptyset$: Finite set of system states;
- $\mathcal{A} \neq \emptyset$: The set of actions that can be executed by the agent;
- $\mathcal{T}$: The state transition function that provides the probability of executing a transition from a state s to a state s' under action a;
- $\mathcal{R}$: The reinforcement function that returns a real value that corresponds to the received reinforcement after each action a at any state s of the environment.

Reinforcement Learning [5], [8] is a collection of learning techniques for MDPs where an agent tries to maximize a function of the total reinforcement values received in a partially or totally unknown environment w.r.t. the transition function. The widely studied Q-Learning algorithm was introduced by Watkins [9] and had its convergence proved in [6]. It is the *de facto* standard RL technique with many variations [10] where the agent learns an optimal (or near-optimal) action policy through sequential updates of an action-value function $Q(s, a)$ without an explicit need to learn a model of the environment. By performing an action a, the agent interacts with the environment moving from state to state. Each action performed over a state provides a reinforcement, and the corresponding agent updates its estimate of the Q-value associated with the state-action pair. In stochastic environments (typical for MDP problems), the Q-value for a given state-action pair under a given action policy is the (temporally discounted) expected sum of the received reinforcement values when performing the action at the given state and following the optimal policy thereafter.

The Q-learning update equation is:

$$Q(s,a) = Q(s,a) + \alpha[R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a)]$$

where $0 < \alpha \leq 1$ is an input learning parameter that determines the extent to which newly acquired information will override old values. The constant $0 \leq \gamma < 1$ is the temporal discount factor, lower values make the agent tends to consider only recent reinforcements for updating the action-value function.

A pseudo-code for Q-Learning is presented in Algorithm 1.

Others Q-learning-based algorithms such as Dyna-Q [11] and SARSA [12], are reportedly faster to converge to the optimal policy than Q-learning. Dyna-Q is an architecture to integrate planning, acting and learning. It uses the same procedure to update the utility values as Q-Learning, however it executes a learning procedure for the expected utility values by looping over an internally updated model of the environment.

Algorithm 1 Q-Learning

```
1: Initialize Q(s, a) arbitrarily;
2: for i = 1 → MAX_EPISODES do
3:    Initialize s
4:    repeat
5:       Choose a from s using policy derived from Q
6:       Take action a, observe R(s) and resulting state s'
7:       Q(s,a) ← Q(s,a) + α[R(s) + γ max_a' Q(s',a') −
         Q(s,a)]
8:       s ← s'
9:    until s is terminal
10: end for
```

SARSA is an acronym for State-Action-Reward-State-Action, due to the procedure executed to update the utility values, where: s is the current state, a is the action chosen to be executed in the state, r the corresponding reward, s' is the resulting state after executing a over s and finally a' is the next action to be executed in s'. The agent interacts with the environment and updates the policy based on taken actions. This technique is an instance of the so-called on-policy learning algorithms.

The above Reinforcement Learning algorithms were compared in [13] and were also used for benchmarking the experiments performed in this paper.

B. Communication

Communication is commonly used to improve the performance of several algorithms in decentralized multi-agent settings. It plays different roles in multi-agent systems, such as information exchange, task delegation, coordination of teams, distributed planning [14], conflict resolution, etc.

In the context herein, the communication between agents [15] is basically the exchange of messages for gathering propositional information about the state space. Although broadcasting models for message difusion are possible, it is generally not interesting to send messages to all agents at each time interval, since communication generally involves some cost [16]. Moreover, broadcasting promotes traffic increase that can cause an overflow [17]. Norrozi [18] shown an efficient way to avoid this problem. In general, for MDP problems the optimal policy for each agent should be generated through minimal and sufficient communication for coordination, such that the cost is not an obstacle as far as costs are concerned. In fact, proper communication can be in some cases crucial for agents to coordinate properly, keeping updated information on the environment. For this, the rules for communication must defined in the model and thus choosing the right moment

to communicate can be considered a fundamental problem in multi-agent systems [19].

C. Extended Reachability Goals

In our approach, the propositional constraints are based on the concept of extended reachability goals. A simple reachability goal corresponds to a condition to be satisfied at the final state in a planning problem, reached after a plan execution [7]. Differently, an extended reachability goal, besides the specification of the condition to be achieved, establishes a condition to be preserved (or restrictions to be satisfied) for every state during the execution. This provides a significant extension on the specification of planning problems, narrowing the scope of the model and allowing more complex planning problems to be more formally defined.

To define extended reachability goals it is necessary to represent them using a formal language, *e.g.* the temporal logic used by Bacchus [20]. In general, the formal representation of the goals is the composition of atomic propositions that represent state characteristics and logical operands, called hereafter expressions. For example, assuming that p, q and k are atomic propositions that correspond to conditions for states, we could define the condition to be preserved as: $\neg p \wedge \neg q$. This representation denotes that states with the properties p or q must be avoided during the interaction with the environment. Similarly, it might be possible to define the condition to be achieved as k, indicating that the problem is solved after reaching a state that satisfies this condition.

Planning problems with extended reachability goals can be solved using a strong probabilistic planning algorithm called PPF' [7]. The inputs to PPF' are $\mathcal{S}, \mathcal{A}, \mathcal{T}$ and the extended reachability goals features. It returns a valid policy, if one exists. If there is more than a single valid policy and the output is the one with the highest probability of reaching the goal. PPF' initially computes a set of states that satisfies the final goal. From this set, the algorithm verifies which of the remaining states that satisfy the preservation goal can reach one state of the set. The action selected has the maximum value for Q, but does not consider the information of the reward function. This step is repeated until the initial state is comprised inside the set or there is no new state to add to the set.

A logic that can be used as a formal language to specify extended reachability goals and a planning system based on this logic is provided by Pereira et al. [7] .

III. RL, MDPs and Extended Reachability Goals

Classical planning and MDP models are not suitable to formally represent extended reachability goals. Hence, a new model is proposed and named ERG-MDP [21], which extends the default MDP model and increases its specificity by adding definitions to manipulate a set of propositions for each state. An ERG-MDP constrains the environment using the extended reachability goals.

The ERG-MDP model contains four additional entities [1] $\mathbb{P}, \mathcal{L}, \varphi_1$ and φ_2, defined as:

- $\mathbb{P} \neq \emptyset$: A non-empty set of atomic propositions representing state characteristics;
- $\mathcal{L} : \mathcal{S} \mapsto 2^{\mathbb{P}}$: The state interpretation function;
- φ_1: The logical expression for the preservation goal to be maintained during the plan execution;
- φ_2: The condition to be achieved at the end of the plan execution defined as a logical expression.

Our proposal is mainly composed by the Multi-ERG-Controller algorithm, which derives from its single agent version, the ERG-Controller [21].

Both the ERG-Controller and the Multi-ERG-Controller make use of two algorithms: PPFERG and ERG-RL. The former is a slightly modified version of the PPF' algorithm that returns *all* the viable policies given the final goal and the preservation goal. The latter incorporates the extended reachability goals features and is detailed as follows.

A. ERG-RL

The ERG-RL algorithm proposed by Araujo and Ribeiro [21] extends a reinforcement learning algorithm to include extended reachability goals. The main difference between ERG-RL and standard RL is that the first also stores the proposition function when interacting with the environment. The expressions are stored together with its corresponding utility value for each state, for posterior evaluation in the ERG-Controller.

We stress that any RL algorithm can be used as the learning component in ERG-RL to update the utility table values. For example, if we use Q-Learning, we have the algorithm ERG-Q, which is considered for the experiments reported in this paper.

B. ERG-Controller

The ERG-Controller algorithm executes the exploration over the environment and plans over the information retrieved from it. The execution flowchart of the algorithm is presented in Figure 1.

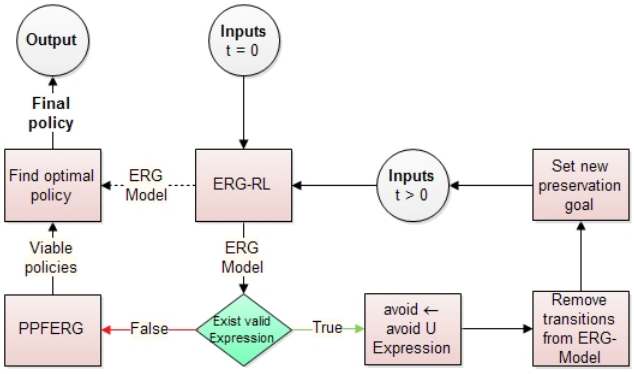

Figure 1. The complete ERG-Controller flowchart.

[1]The default MDP properties that were previously defined in section II are omitted.

The algorithm initially explores the environment to generate a first ERG-Model. This occurs through the execution of the ERG-RL algorithm.

After a ERG-RL execution, the ERG-Controller tries to find the expression with the lowest utility value for all visited states which is not in the set of avoidable expressions. If the algorithm finds the expression and it does not obstruct the agent from reaching the final goal, then it removes all the transitions that directs to states that contains the expression found. Afterwards, it continues the execution while accumulating experiences to improve the model.

After the main loop, the PPFERG algorithm runs over a state transition model generated by bootstrapping the experiences from an exploratory action policy of ERG-RL. The execution of PPFERG guarantees that the expression defined by the preservation goal is valid for all possible states and directs the actions towards the states that satisfy the final goal. By establishing these conditions, PPFERG implicitly reduces the state space.

Finally, the ERG-Controller returns a policy that corresponds to an optimal policy based on the set of viable policies found by the PPFERG algorithm. This optimal policy corresponds to executing the actions that produce the maximum utility values for the corresponding states.

A more detailed description of ERG-RL can be found in [21].

C. Multi-ERG-Controller

The ERG-controller does not support multi-agent executions for simultaneous update of the ERG-Model. To apply the same approach to problems with different agents distributed over the environment, we extended the default ERG-Controller and named it Multi-ERG-Controller.

The Multi-ERG-Controller executes a different thread for each agent in the environment. Each thread executes an ERG-RL algorithm instance according to the agent's initial position. Figure 2 presents the flowchart for the Multi-ERG-Controller algorithm.

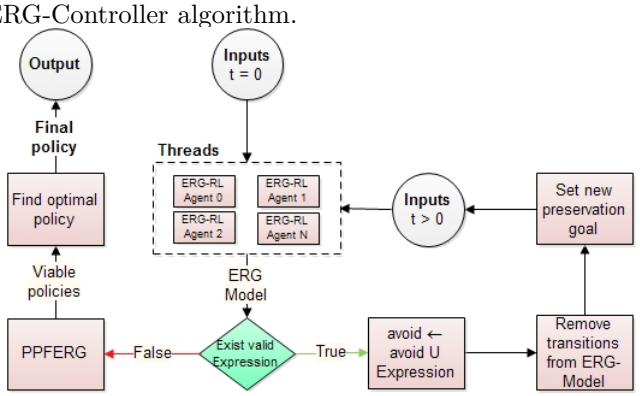

Figure 2. The synthesized Multi-ERG-Controller approach flowchart.

The flowchart in Figure 2 shows the execution of a number of threads equal to the number of agents, which is its main feature. The detailed Multi-ERG-Controller algorithm is presented in Algorithm 2.

Algorithm 2 Multi-ERG-Controller

Require: θ

1: $avoid \leftarrow \neg\varphi_1$
2: **repeat**
3: $U \leftarrow$ U-Values(RunRLThreads($\mathcal{E}$))
4: $\mathcal{E} \leftarrow \mathcal{E} \cup model$(RunRLThreads($\mathcal{E}$))
5: **for all** $s \in \mathcal{S}$ **do**
6: $exp \leftarrow \mathcal{L}(s)$
7: **if** $exp \notin avoid \wedge \neg blocked(exp, \varphi_2)$ **then**
8: $VE \leftarrow VE \cup \{exp\}$
9: $count(exp) \leftarrow count(exp) + 1$
10: $sum(exp) \leftarrow sum(exp) + U(s)$
11: **end if**
12: **end for**
13: **if** $VE \neq \emptyset$ **then**
14: **for all** $exp \in VE$ **do**
15: **if** $sum(exp) \div count(exp) \leq \theta$ **then**
16: $mean(exp) \leftarrow sum(exp) \div count(exp)$
17: **end if**
18: **end for**
19: $exp \leftarrow \min_{mean}$
20: **if** $exp \neq null$ **then**
21: $avoid \leftarrow avoid \cup \{exp\}$
22: $\mathcal{T} \leftarrow \mathcal{T} - \{\forall_s \in \mathcal{S}, \forall_a \in \mathcal{A}, \mathcal{T}(s,a) \rightarrow s'$ where $exp \in \mathcal{L}(s')\}$
23: **end if**
24: **end if**
25: **until** $VE = \emptyset$
26: $\varphi_1 \leftarrow \neg avoid$
27: $\mathcal{P} \leftarrow PPFERG(\mathcal{E})$
28: $\pi \leftarrow findOptimal(\mathcal{P}, U)$
29: **return** π

Both the U-Values, U in algorithm 2, and the ERG-Model ($\mathcal{E}$) are extracted from the agents' interaction with the environment. These interactions are executed in the procedure RunRLThread (detailed in Algorithm 3). U represents the utility values and $\mathcal{E}$ the model retrieved from the inner RL algorithm.

The acquired U-Values and the ERG-Model are used in the Multi-ERG-Controller to decide which expression (exp) with averaged utility value below θ [2] that does not block the final goal (φ_2) must be avoided. Here, one expression is composed by the group of propositions present in a state joined by an $and(\wedge)$ operator. The propositions comprised in a state can be recovered through the state interpretation function ($\mathcal{L}$).

After each round of the main loop, the algorithm verifies if there is a valid expression. If a valid expression exists, the algorithm chooses the expression, stores it in the set of avoidable expressions ($avoid$) and removes the transitions that reach states that represents the expression. If there

[2]Input parameter that defines the minimum utility value below which an expression must be avoided during the interaction.

is not a valid expression, it calls the PPFERG algorithm, that operates similarly in the ERG-Controller.

As in the ERG-Controller, the algorithm then returns an optimal policy based on the set of viable policies (π) found by PPFERG.

RunRLThreads is responsible for creating and executing an ERG-RL instance asynchronously for each agent in the problem. This algorithm returns the U-Values extracted from the interaction with the environment and the ERG-MDP model ($\mathcal{E}$). Figure 3 shows a graphical scheme for it, and Algorithm 3 details its operation.

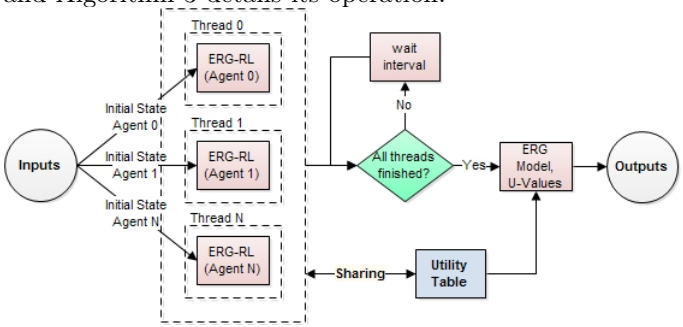

Figure 3. Graphical scheme for the RunRLThreads algorithm.

Algorithm 3 RunRLThreads

Require: initial-states

1: **for all** *state* ∈ initial-states **do**
2: $Thread$(ERG-RL(*state*))
3: **end for**
4: **repeat**
5: wait-time-interval
6: **until** all threads have finished.
7: **return** U-Values, $\mathcal{E}$

Algorithm 3 calls the procedure *Thread*, which is responsible for creating a new execution process and scheduling it in the operating system to run. Once all threads are running, the algorithm has to wait a predefined time interval (represented by the procedure wait-time-interval) to verify if all threads have finished. After all threads have been executed, the algorithm can finally return the updated utility table and the model extracted from the RL executions. Information exchange among the agents is accomplished by sharing the same utility table (U-Values). Thus, all information gathered from the environment is updated in the same table. After each update, all agents can retrieve the current utility value for each state, bit to avoid inconsistency errors, the access to the utility table is synchronized. The final policy is retrieved from the shared utility table by the Multi-ERG-Controller, which selects the actions with the maximum utility value for each state.

IV. Experiments

The experiments were ran on grid environments with 100 states, 10 rows and 10 columns, where each cell corresponds to a single state. They were defined with extended reachability goals and the valid propositions are: A, B and @, the latter representing the final goal to be reached. Table I summarizes the abbreviations and descriptions for the propositions and possible actions.

The grid environments were randomly generated w.r.t position of the agents, obstacles, initial positions and final goal position.

Table I
Abbreviations and descriptions for the generated examples.

Abbrev.	Name	Description
0-9	Agents	Agents' Identifiers
A	Obstacle A	Proposition with low reward
B	Obstacle B	Proposition with low reward
@	Final Goal	Composition of the final goal
↑	North	Execute the action "north"
↓	South	Execute the action "south"
→	East	Execute the action "east"
←	West	Execute the action "west"

Figure 4 illustrates an environment configuration (scenario) generated according to the description in Table I). Empty cells correspond to states that do not have any proposition associated.

	0	1	2	3	4	5	6	7	8	9
0	0		A B			@				
1	1			A B						2
2			A B			A B				B
3					B		A B			A B
4		A								
5	A	A							B	A
6										
7	B	B					A	A		
8					A B		B	A		A
9						B				A B

Figure 4. An example of generated environment.

The ERG-MDP model $\mathcal{E}$ was defined as follows:

- $\mathcal{S}$: $[s00, ..., s99]$;
- $\mathcal{A}$: $[north, south, west, east]$;
- $\mathbb{P}$: $[A, B, @]$;
- $\mathcal{T}$: Equally distributed between the possible states[3];
- $\mathcal{L}$: It was randomly created for each different example;
- $\mathcal{R}$[4]:
  - $\forall_s \in \mathcal{S}, A \vee B \in \mathcal{L}(s), R(s) : -30$;
  - $\forall_s \in \mathcal{S}, @ \in \mathcal{L}(s), R(s) : 30$;
  - $R(*) : -1.0$.
- φ_1: $\emptyset$;
- φ_2: @.

The experiments were performed on a set of 10 examples performed 10 times for each test case. For these experiments, the Multi-ERG-Controller algorithm was executed with 3 agents (one thread each). The ERG-Controller and the RL algorithms Q-learning, SARSA and Dyna-Q were

[3]e.g. if agent is in state s00 there are only two possible actions: south and east, each one with probability 0.5.

[4]where the symbol '*' means any action or state.

also executed over the same test cases for comparative purposes.

For operating the RL algorithms, the required constants were defined with teh following values: $\alpha = 0.1, \gamma = 0.9$ and $\epsilon = 0.09$. For the ERG-Controller based algorithms, $\theta = -15$. The execution of Dyna-Q requires also the parameter N, which is the number of iterations for the internal planning procedure, defined as 5.

The stopping criterion was defined as $\max Q_{t-1}(s,a) - Q_t(s,a) < \epsilon$, for all executions.

V. Results

This section presents the final preservation goal achieved by the ERG controllers and the comparison analysis between Multi-ERG-Controller (with 3 agents), ERG-Controller, Q-Learning, Dyna-Q and SARSA.

A. Preservation Goal

At each iteration of the Multi-ERG-Controller and ERG-Controller main loops, the algorithms find the expression with the lowest averaged utility value, until it finds all suited expressions (according to θ). The resulting set of expressions is composed by: A, B and $A \wedge B$. Thus, the final preservation goal found by both ERG based algorithms is:

$\neg(A \vee B \vee (A \wedge B))$

The preservation goal indicates that the agent should avoid the states that contains expressions with one of the propositions (A or B) or both.

B. Executions

Table II shows the overall results considering Q-Learning, SARSA and Dyna-Q as benchmark RL algorithms against the employment of them as the learning component of ERG-RL in the ERG based algorithms. The total time (T) of an execution (in milliseconds), the number of iterations and the standard deviation of the number of iterations were considered in the analysis.

The stopping criterion for all RL algorithms consists in comparing the V values obtained via Policy Evaluation and the optimal V* values obtained via the Value Iteration (VI) algorithm [22]. The ERG based algorithms have to rerun Value Iteration after each discovery of a new expression to be avoided, because as some states get blocked, the environment changes and correspondingly its V* values. The time of running Value Iteration is therefore discounted from the total time of execution for every algorithm. The average Value Iteration execution time for all executions was 730.46 ms. In our experiments there are 3 avoided expressions that are found, so the total amount of time to be discounted is 2191.38 ms.

For the Multi-ERG-Controller executions, all the compared values are the average of the results for all agents.

Table III presents the percentage of improvement when comparing the ERG-Controller against the corresponding standard RL algorithms. Table IV compares the single agent versions of the ERG-Controller with its corresponding multi-agent versions, showing that there was a reduction of time of execution and number of episodes for reaching the stopping criterion in the multi-agent instances.

Table II
Comparative results between all tested algorithms.

Algorithm	T	Iterations	Iter. (std dev)
Q-Learning	3370	30.1	8.03
ERG-Cont. (Q-Learn.)	2225.6	18.1	5.46
Multi-ERG (Q-Learn.)	1709.6	7.9	4.1
SARSA	3655	21.7	16.87
ERG-Cont. (SARSA)	2955.6	14.3	3.05
Multi-ERG (SARSA)	2404.6	7.2	3.47
Dyna-Q	2088	13.6	8.3
ERG-Cont. (Dyna-Q)	1595.6	9.0	2.26
Multi-ERG (Dyna-Q)	1044.6	3.66	1.72

Table III
Comparison, RL algorithms and its ERG-RL versions.

Algorithm	$\%T$	% Iterations	Iter. (std dev)
Q-Learn	33.95	39.86	32.0
SARSA	19.13	34.10	81.92
Dyna-Q	23.58	33.82	72.77

Table IV
Comparison, ERG-RL algorithms and its Multi-ERG-RL versions.

Algorithm	$\%T$	% Iterations	Iter. (std dev)
Q-Learn	23.18	56.35	24.90
SARSA	18.64	49.65	-13.77
Dyna-Q	34.53	59.33	23.89

Finally, Table V shows the comparison of the standard RL algorithms with the Multi-ERG-Controllers that use the corresponding RL algorithms, showing that there was a remarkable reduction of time of execution and number of episodes for reaching the stopping criterion in the multi-agent instances.

Table V
Comparison between the standard RL algorithms and its Multi-ERG-RL versions.

Algorithm	$\%T$	% Iterations	Iter. (std dev)
Q-Learn.	49,27	73.75	48.94
SARSA	34.21	66.82	79.43
Dyna-Q	49.97	73.08	79.27

VI. Conclusion

In this paper, we presented a new approach, called Multi-ERG-Controller, to discover the action policy for an environment under propositional constraints on states in MDP problems for multiple agents. Our method connects extended reachability goals and multiple reinforcement learning instances through shared information. We implemented a shared utility table algorithm, according to the significance of environmental observations made by the agents, in order to reduce the learning time and generate better action policies.

The results obtained and presented in section V showed an important reduction of learning time and iterations when applying any ERG-based algorithm, this reduction is more remarkable in the Multi-ERG-Controllers, which encourages its use in problem solving with extended reachability goals for multiple agents.

As future work, we intend to adapt the Multi-ERG-Controller to partially observable MDPs and extended reachability goals.

References

[1] L. Panait and S. Luke, "Cooperative multi-agent learning: The state of the art," *Autonomous Agents and Multi-Agent Systems*, vol. 11, pp. 387–434, 2005, 10.1007/s10458-005-2631-2. [Online]. Available: http://dx.doi.org/10.1007/s10458-005-2631-2

[2] R. A. Howard, *Dynamic Programming and Markov Processes*. MIT Press and Wiley, 1960, vol. 3.

[3] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, 1st ed. New York, NY, USA: John Wiley & Sons, Inc., 1994.

[4] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Pearson Education, 2003. [Online]. Available: http://portal.acm.org/citation.cfm?id=773294

[5] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*. The MIT Press, mar 1998.

[6] C. J. C. H. Watkins and P. Dayan, "Technical note q-learning," *Machine Learning*, vol. 8, pp. 279–292, 1992.

[7] S. Lago Pereira, L. Barros, and F. Cozman, "Strong probabilistic planning," in *MICAI 2008: Advances in Artificial Intelligence*, ser. Lecture Notes in Computer Science, A. Gelbukh and E. Morales, Eds. Springer Berlin / Heidelberg, 2008, vol. 5317, pp. 636–652, 10.1007/978-3-540-88636-5_61. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-88636-5_61

[8] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: a survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.

[9] C. Watkins, "Learning from delayed rewards," Ph.D. dissertation, University of Cambridge,England, 1989.

[10] A. L. Strehl, L. Li, E. Wiewiora, J. Langford, and M. L. Littman, "Pac model-free reinforcement learning," in *Proceedings of the 23rd international conference on Machine learning*, ser. ICML '06. New York, NY, USA: ACM, 2006, pp. 881–888. [Online]. Available: http://doi.acm.org/10.1145/1143844.1143955

[11] R. S. Sutton, "Integrated architectures for learning, planning, and reacting based on approximating dynamic programming," in *Proceedings of the Seventh International Conference on Machine Learning*. Morgan Kaufmann Publishers Inc., 1990, pp. 216–224.

[12] G. A. Rummery and M. Niranjan, "On-line Q-learning using connectionist systems," Cambridge University Engineering Department, Cambridge, England, Tech. Rep. TR 166, 1994.

[13] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," in *Machine Learning*, 1992, pp. 293–321.

[14] E. H. Durfee and V. R. Lesser, "Using partial global plans to coordinate distributed problem solvers," in *Proceedings of the 10th international joint conference on Artificial intelligence - Volume 2*. San Francisco, CA, USA: Morgan Kaufmann, 1987, pp. 875–883. [Online]. Available: http://portal.acm.org/citation.cfm?id=1625995.1626060

[15] C. V. Goldman and S. Zilberstein, "Optimizing information exchange in cooperative multi-agent systems," in *AAMAS '03: Proceedings of the second international joint conference on Autonomous agents and multiagent systems*. New York, NY, USA: ACM, 2003, pp. 137–144.

[16] P. Xuan, V. Lesser, and S. Zilberstein, "Communication in multi-agent markov decision processes," in *In Proc. of ICMAS Workshop on Game Theoretic and Decision Theoretic Agents*, 2000.

[17] M. Kinney and C. Tsatsoulis, "Learning communication strategies in multiagent systems," in *In Applied Intelligence*, 1998.

[18] A. Noroozi, "A novel model for multi-agent systems to improve communication efficiency," in *Computer Engineering and Technology, 2009. ICCET '09. International Conference on*, vol. 2, jan. 2009, pp. 189 –192.

[19] R. Becker, A. Carlin, V. Lesser, and S. Zilberstein, "Analyzing myopic approaches for multi-agent communication," *Computational Intelligence*, vol. 25, no. 1, pp. 31–50, 2009. [Online]. Available: http://dx.doi.org/10.1111/j.1467-8640.2008.01329.x

[20] F. Bacchus and F. Kabanza, "Planning for temporally extended goals," *Annals of Mathematics and Artificial Intelligence*, vol. 22, pp. 5–27, 1998, 10.1023/A:1018985923441. [Online]. Available: http://dx.doi.org/10.1023/A:1018985923441

[21] A. V. Araujo and C. H. C. Ribeiro, "Solving problems with extended reachability goals through reinforcement learning on propositionally constrained state spaces," in *IEEE International Conference On Systems, Man, And Cybernetics*, vol. 1, 2013, pp. 1542–1547.

[22] R. Bellman, *Dynamic Programming*, 1st ed. Princeton, NJ, USA: Princeton University Press, 1957.