A New Distributed Reinforcement Learning Algorithm for Multiple Objective Optimization Problems

Carlos E, Mariano CMARIANO@TLALOC.IMTA.MX
Instituto Mexicano de Tecnología del Agua, Paseo Cuauhnáhuac 8532, Jiutepec, Morelos 62550 México

Eduardo Morales EMORALES@CAMPUS.MOR.ITESM.MX
ITESM - Campus Morelos, Paseo de la Reforma 182-A, Temixco, Morelos 62589 México

Abstract

This paper describes a new algorithm, called MDQL, for the solution of multiple objective optimization problems. MDQL is based on a new distributed Q-learning algorithm, called DQL, which is also introduced in this paper. In DQL a family of independent agents, exploring different options, finds a common policy in a common environment. Information about action goodness is transmitted using traces over state-action pairs. MDQL extends this idea for multiple objectives, assigning a family of agents for each objective involved. A non-dominant criterion is used to construct Pareto fronts and by delaying adjustments on the rewards MDQL achieves better distributions of solutions. Furthermore, an extension for applying reinforcement learning to continuos functions is also given. Successful results of MDQL on several test-bed problems suggested in the literature are described.

1. Introduction

Multiobjective optimization is with no doubt a very important research topic for scientist and engineers, not only because of the multiobjective nature of most real problems, but also because there are still many open questions in this area. The multiobjective optimization problem is, without loss of generality, the problem of simultaneously minimizing the n components $f_k, k = 1, \ldots, n$, of a vector function $\mathbf{f}$ of a variable $\mathbf{x}$ in a universe u, where: $\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), \ldots, f_n(\mathbf{x}))$.

The problem has usually no unique, perfect solution, but a set of equally efficient, or non-inferior, alternative solutions, known as the Pareto-optimal set. Multiobjective optimization using genetic algorithms has been investigated by many authors in recent years e.g., (Fonseca & Flemming, 1995a; Shaffer, 1985; Valenzuela & Uresti, 1997; Veldhuizen, 1999; Zitzler & Thiele, 1998), among others. However in some real-world optimization problems the application of genetic algorithms is limited by the requirement of finding an adequate codification of the problem. Additionally the parameters associated with the evolutionary strategies, such as: weight factors used in function aggregation, priorities used in lexicographic ordering and Pareto ranking, niche size in niche induction techniques, tournament size, etc. (e.g., see (Fonseca, 1995a)), need to be established. Finding the appropriate values for the parameters in evolutionary algorithms is most of the time as difficult as the main problem solution.

In Operations Research, many techniques (more that 20 (Coello, 1999)), have been developed over the years to try to deal with functions that have multiple objectives. They range from the combination of objectives into a single one to the use of game theory to coordinate the relative importance of each objective. However, and because there is no accepted definition of "optimum" as in single-objective optimization, it is difficult to even compare results of one method to another, and the decision about what is the "best" answer corresponds to the, so called, *decision maker*.

Reinforcement learning (RL) has recently received considerable attention. In particular, in RL an autonomous agent learns an optimal policy that outputs an appropriate action given the current state (Sutton, 1998) and can be easily formulated to solve optimization problems. The first proposed algorithm, to our knowledge, to solve multiple objective optimization problems based on RL was MOAQ (Mariano & Morales, 1999a; Mariano & Morales, 1999b). MOAQ can be seen as a natural extension of Ant-Q (Gambardella & Dorigo, 1995) for solving multiple objective optimization problems. It suffers, however, from similar limitations as Ant-Q, namely, it assumes that a domain dependent heuristic is available and that a suitable combination with the evaluation function can be determined. It also requires the definition of extra parameters which need to be tuned for each particular problem and looses all the theoretical results from reinforcement learning. Despite all these, MOAQ was used to successfully solve a highly restricted and complex multiple objective optimization problem (Mariano & Morales, 1999a) and other problems reported in the genetic algorithms community (Mariano & Morales, 1999b).

This paper first introduces a new approach to Multi-agent Reinforcement Learning designed to solve problems with a common task. In this algorithm, called DQL, agents use "traces" of all the other agents, generated from transitions between states, to infer a policy based on the most frequently selected actions. A new algorithm, called MDQL, is then presented for the solution of multiple objective optimization problems. It uses a family of agents for each objective involved and a negotiation mechanism based on non-dominance to produce compromise solutions. A further extension for the optimization of real-valued functions is also introduced. It is shown how MDQL is able to solve several multiple objective optimization problems suggested in the literature, outperforming the best genetic algorithm reported in (Veldhuizen & Lamont, 1999).

2. Reinforcement Learning

In reinforcement learning, an autonomous agent learns an optimal policy $\pi :\rightarrow A$, that outputs an appropriate action $a \in A$, given the current state $s \in S$, where A is the set of all possible actions in a state and S is the set of states. The available information to the agent is the sequence of immediate rewards $r(s_i, a_i)$ for all the possible actions and states $i = 0, 1, 2, \ldots$. Given this kind of training information it is possible to learn a numerical evaluation function defined over states and actions, and then implement the optimal policy in terms of this evaluation function. The value of the evaluation function $Q(s, a)$ is the expected reward when starting in state s, taking action a, and thereafter following policy π.

One of the most important breakthroughs in reinforcement learning was the development of the temporal difference algorithm known as *Q-learning* (Watkins, 1989). Its simplest form, one step Q-learning, is defined by:

$$\begin{aligned} Q(s,a) \quad \leftarrow \quad & Q(s,a) + \\ & \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s,a) \right] \end{aligned} \tag{1}$$

where, $Q(s, a)$ is the action value function of taking action a in state s, $Q(s', a')$ is the action value function in the resulting state (s') after taking action a, α is the step-size parameter $[0, 1]$, r is the reward, and γ is a discount rate. The action value function is updated using information from the current value, the associated reward, and the best next action value function.

Q-Learning and other RL algorithms have been reported in the literature for multiple agent settings, e.g., (Boutilier, 1996; Claus & Boutilier, 1998; Hu & Wellman, 1998; Littman, 1994; Tan, 1993). They can be characterized by the way the agents interact with each other. For instance, agents knowing what the other agents perceive (Tan, 1993), agents that share the learned policies (Tan, 1993), agents that select actions based on the estimation of dynamic probabilities (Boutilier, 1999, Claus & Boutilier, 1998), the establishment of optimal policies for a certain environment (Littman, 1994), or the establishment of updating rules for state values based on strategies that mix the learned policies (Hu & Wellman, 1998), among others.

Most the reported work is focused on the selection of rules for the coordination of actions between agents. In the following section, a simple approach for the coordination of agents searching for a common goal, called DQL, is described.

2.1 Distributed Q-Learning

In DQL, a set or family of independent agents is trying to find an optimal policy in a common environment. In this setting, all the agents are exploring different options, all have the same goals, and all use the same policy, which is updated by all agents. Information transmission about action goodness is given by the information of traces over state-action pairs. The updating of each trace value depends on the current state-action value and on the value of the next time step state-action pair to consider[1]:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[\gamma \max_{a'} Q(s', a') - Q(s,a) \right] \tag{2}$$

Initial $Q(s, a)$ values are arbitrarily selected. Each agent starts an episode on an arbitrarily selected state.

[1]A greedy policy is used, although other policies are possible.

Table 1. General algorithm for DQL.

DQL
Initialize $Q(s,a)$ arbitrarily
Repeat (for n episodes)
Repeat (for m agents)
Initialize s, copy $Q(s,a)$ to $Q_C(s,a)$
Repeat (for each step of episode)
Take action a, observe r, s'
$Q_C(s,a) \leftarrow Q_C(s,a) +$
$\alpha \left[\gamma \max_{a'} Q_C(s',a') - Q_C(s,a)\right]$
$s \leftarrow s'$;
Until s is terminal
Evaluate the m proposed solutions
Assign rewards to the best solution found
$Q(s,a) \leftarrow Q(s,a) +$
$\alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$

An episode for an agent ends when it reaches a terminal state. Before starting an episode, all the Q values are copied. On each state s, an agent greedily selects an action a which takes it to a state s' with the maximum estimated $Q(s',a')$ and updates its Q value (copy) according to equation (2). Frequently visited state-action pairs end with more "traces" than less frequently visited pairs. Once all the agents have finished their episodes, all the proposed solutions are evaluated and the best one is rewarded with a numerical value, r, according to equation (1). This updating is performed over the original Q values and the whole cycle is repeated several times until reaching a termination criterion (see Table 1).

DQL's behavior is illustrated on a simple maze problem with a wind force in the middle of the maze (see Fig. 1). The allowed standard moves are *up, down, right*, and *left*. In the central region, the resulting states after an action are affected by the wind force moving the expected state one or two cells above. The amount of displacement is shown on the horizontal axis of Figure 1. This problem is considered an episodic task without discounts and constant rewards of -1.0 until reaching the goal, whore the reward is equal to $+1.0$. Figures 2 and 3 show the best results of 10 runs using Q-learning (1 agent) and DQL with 3 and 10 agents (Figure 3).

As can be appreciated from the figures, the number of iterations is reduced when increasing the number of agents to 1/3 for 3 agents and 1/4 for 10 agents. In all cases the following parameters were used: $\alpha = 0.1, \gamma = 1$, and $Q(s,a)$ initial values set to 0. An ε-*greedy* policy was used to select a (with $\varepsilon = 0.1$) and a *greedy* policy to select a'. All algorithms converge to the same solution.

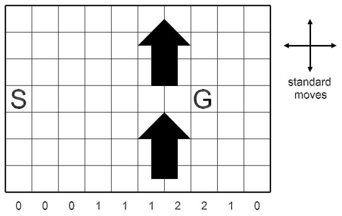

Figure 1. Gridworld in which movement is altered by a location-dependent, upward "wind."

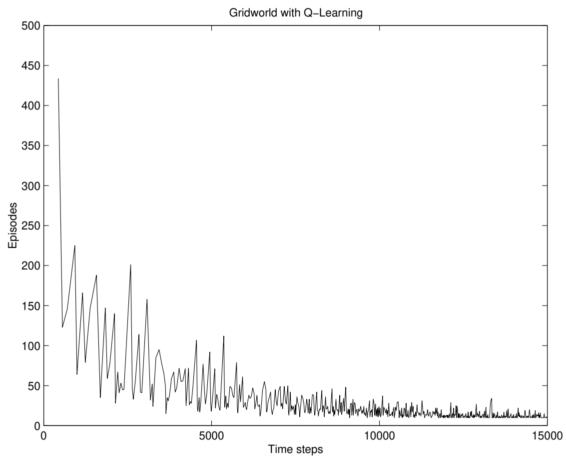

Figure 2. Results of Q-Learning applied to the windy gridworld.

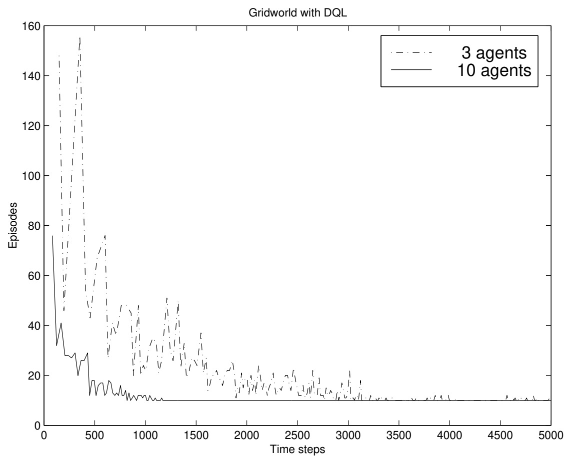

Figure 3. Results of DQL applied to the windy gridworld, 3 and 10 agents were considered for the solution.

Table 2. General algorithm for MDQL, the same number of agents is considered for all families.

MDQL
Initialize $Q(s,a)$ arbitrarily
Repeat (for n episodes)
Initialize s, copy $Q(s,a)$ to $Q_C(s,a)$
Repeat (for m agents)
Repeat (for f families)
Repeat (for each step of episode)
Take action a, observe r, s'
$Q_C(s,a) \leftarrow Q_C(s,a)+$
$\alpha\left[\gamma \max_{a'} Q_C(s',a') - Q_C(s,a)\right]$
$s \leftarrow s'$;
Until s is terminal
compromise_solution=**negotiation**(m solutions)
if (*compromise_solution* are *non-dominated*)
Pareto=Pareto + *compromise_solution*
For all Pareto solutions do
$Q(s,a) \leftarrow Q(s,a)+$
$\alpha\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$

2.2 Distributed Q-Learning for Multiobjective Optimization Problems

DQL can be extended for multiple objective optimization problems. The idea is to assign a family of agents to each objective. The solutions obtained by the agents in one family are compared with the solutions obtained by the agents from the rest of the families. A negotiation mechanism is used to find compromise solutions satisfying all the objectives. This mechanism was originally proposed and tested in (Mariano, 1999a) and (Mariano, 1999b) using Ant-Q (Gambardella & Dorigo, 1995). The limitations and extra parameters associated with Ant-Q based algorithms lead us to propose a more theoretical sustained algorithm based exclusively on Q-learning. This algorithm, called MDQL, is described in Table 2.

The associated rewards of the states involved in the solutions are updated according to their evaluation function (this is described in the following section). By delaying the adjustment in the rewards MDQL achieves a better distribution of solutions (Pareto set) while keeping its exploration capabilities.

In the negotiation mechanism all solutions are evaluated for all the objectives. If a solution satisfies all the objectives it is considered as a candidate compromise solution. If there is more than one possible solution, the first one is returned. MDQL follows a non-dominant criterion to construct a Pareto front. Non-dominated solutions are kept in the Pareto front and rewards are assigned to their components. In cases

Table 3. Negotiation mechanism that compares the solution proposed by an agent with the solution proposed by agents from other families.

Negotiation(m solutions)
Let *compromise_solution*=NULL
Repeat (for each solution i)
Repeat (for each objective j)
if (objective$_j$(solution$_i$) achieves
all objective$_j$(solution$_j$))
compromise_solution=
best_of(solution$_i$, *compromise_solution*)
if(*compromise_solution*=NULL)
compromise_solution=**random**(m solutions)
return *compromise_solution*

where there is no solution which dominates the rest, a new solution is randomly selected from the rest of the solutions at that episode. The state-action evaluation functions are updated as in DQL. The number of episodes and steps are given arbitrarily, as long as a sufficiently large number is chosen to ensure convergence.

2.2.1 Application to optimization problems and continuos functions

To solve optimization problems with continuos functions we need to: (i) discretize the space into a set of states, (ii) decide how to traverse such space, and (iii) establish a criterion for assigning rewards even if we do not know where the goal is. In this paper, we take ideas from George Bilchev & Ian Parmee (1995) and Deniz Yuret (1994) to deal with these problems. George Bilchev and Ian Parmee (1995) proposed an adaptation of the Ant Colony model to continuos problems. His proposal considers for each state a set of possible directions for the trajectory of agents. The magnitude of this trajectory is bounded by an upper limit. At the beginning of the algorithm, the trajectories and magnitudes are randomly selected, and subsequently, the most frequently selected directions are assigned higher probabilities of selection. In an earlier work, Deniz Yuret (1994) proposed an algorithm, named Dynamic Hill Climbing (DHC), which used a similar mechanism. DHC is based on three main heuristics: storage of local minimum and restarts from distant points from those points, adjustment of the magnitudes of the trajectories used during the search, and use of successful trajectories for faster climbings (descendings). Following a similar approach, in our proposal each state has a set of possible actions which depends on the problem to be solved. The actions are associated with the directions that an agent can take, which depends on the

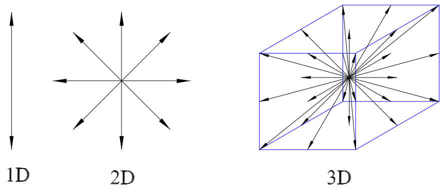

Figure 4. The number of combinations of main directions increases exponentially with the number of directions.

dimensionality of the problem to be solved. For one dimension there are two possible directions, for 2 there are 8 (considering both directions for each dimension and all the directions of the combinations of them), for 3 there are 26, and so on (see Figure 4).

The magnitudes of the trajectories are adjusted according to their evaluation. If a state-action pair increases the evaluation function, its magnitude is doubled, otherwise it is divided by 2. The initial magnitudes are given arbitrarily. In our setting, the magnitude of the trajectories represent the reward that the agent receives when selecting a state-action pair (r in equation (1)). This adjustment on the value of r is used to guide an agent towards the solution of optimization problems where the goal state is not known in advance.

3. Experiments

MDQL was tested on four distinctive multiple objective optimization problems suggested in the literature (see also (Veldhuizen & Lamont, 1999)).

Shaffer's two-objective function was selected because: (i) almost all Multiple Objective Evolutionary Algorithm proposed have been tested using this function, (ii) an analytical expresion of the true Pareto front (PF_{true}) (Veldhuizen & Lamont, 1998) is known, and (iii) it is easy to determine solutions' membership in the true Pareto set (P_{true}). Its PF_{true} is a single convex curve and its P_{true} a line. We rename this function *MOP1*.

$$\begin{array}{l} \textit{minimize } f_1(x) = x^2, \\ \textit{minimize } f_2(x) = (x-2)^2 \\ \textit{subject to: } -10^5 \leq x \leq 10^5 \end{array}$$

Fonseca's second problem was also selected (Fonseca & Flemming, 1995b). The PF_{true}'s shape and location in the objectives space does not change when decision variables are added. Its PF_{true} is a concave Pareto curve and its P_{true} is an area in the solution space. We rename this function *MOP2*.

$$\begin{array}{rcl} \textit{minimize } f(\overrightarrow{x}) & = & 1-\exp(-\sum_{i=1}^{n}(x_i - \frac{1}{\sqrt{n}})^2), \\ \textit{minimize } f_2(\overrightarrow{x}) & = & 1-\exp(-\sum_{i=1}^{n}(x_i + \frac{1}{\sqrt{n}})^2) \\ \textit{subject to: } & & -4 \leq x_i \leq 4,\ i=1,2,3 \end{array}$$

MDQL was also tested on Viennet's third three-objective function (Viennet et al., 1996). Its PF_{true} is a single, convoluted three-dimensional Pareto curve and its P_{true} consists of disconnected areas in the solution space. We rename this function *MOP3*.

$$\begin{array}{rcl} \textit{minimize } f_1(x,y) & = & 0.5(x^2+y^2) + \\ & & \sin(x^2+y^2), \\ \textit{minimize } f_2(x,y) & = & \frac{(3x-2y+4)^2}{8} + \\ & & \frac{(x-y+1)^2}{27} + 15, \\ \textit{minimize } f_3(x,y) & = & \frac{1}{(x^2+y^2+1)} - \\ & & 1.1e^{(-x^2-y^2)} \\ \textit{subject to: } & & -30 \leq x,y \leq 30 \end{array}$$

Finally a problem proposed in (Deb, 1998) was selected. Their P_{true} and PF_{true} (four Pareto curves) are disconnected. We rename this function *MOP4*.

$$\begin{array}{rcl} \textit{minimize } f_1(x,y) & = & x, \\ \textit{minimize } f_2(x,y) & = & (1+10y)[1-\left(\frac{x}{1+10y}\right)^{\alpha} - \\ & & \frac{x}{1+10y}\sin(2\pi q x)] \\ \textit{subject to: } & & 0 \leq x,y \leq 1, q=4,\ \alpha=2 \end{array}$$

MDQL performance was measure with two metrics proposed in the GA literature, in particular, the *error ratio* and *spacing* metrics were used.

Error Ratio: The results reported by the algorithm are in the PF_{MDQL}. Solutions in PF_{MDQL} which are not members of PF_{true} are considered as errors. This metric is mathematically represented by (Veldhuizen & Lamont, 1999):

$$E \triangleq \frac{\sum_{i=1}^{n} e_i}{n}$$

Table 4. Parameters used by MDQL for the four test problems.

	Episodes	Steps	States	Agents	Time[2] (sec.)
MOP1	200	100	240	100	50
MOP2	300	400	800	100	85
MOP3	500	500	500	100	200
MOP4	500	500	2000	150	250

where n is the number of vectors in PF_{MDQL} and

$$e_i = \begin{cases} 0 \text{ if vector } i \in PF_{true} \\ 1 \text{ otherwise} \end{cases}$$

For example, $E = 0$ indicates that every vector reported by MDQL in PF_{MDQL} is actually in PF_{true}; $E = 1$ indicates that none are.

Spacing: To measure the spread (distribution) of vectors throughout PF_{MDQL} a metric proposed in (Schott, 1995) was implemented. This metric measures the range (distance) variance of neighboring vectors in PF_{MDQL}:

$$S \triangleq \sqrt{\frac{1}{n-1}\sum_{i+1}^{n}\left(\overline{d} - d_i\right)^2},$$

where
$d_i = \min_j \left(\mid f_1^i(\overrightarrow{x}) - f_1^j(\overrightarrow{x}) \mid + \mid f_2^i(\overrightarrow{x}) - f_2^j(\overrightarrow{x}) \mid \right)$, $i, j = 1, \ldots, n$, $\overrightarrow{d}$ is the mean of all d_i, and n is the number of vectors in PF_{MDQL}. $S = 0$ indicates all members of PF_{MDQL} are equidistantly spaced.

4. Results

Figures 5 through 8 show the best MDQL solutions for the four problems. All tests were run with $\alpha = 0.1$ and $\lambda = 1$ and initial states at the origin. Table 4 shows the number of episodes, steps per episode, number of states, number of agents and convergence times for each problem. All four functions were run 10 times and compared on both metrics against MOMGA, the best evaluated algorithm in (Veldhuizen, 1999). Table 5 shows the best and worst error-ratio (ϵ_{ratio}) and spacing (σ) metrics of MDQL and MOMGA for the 10 runs.

As can be seen from the results, MDQL is able, in general, to successfully solved a wide range of multiple objective optimization problems. It also behaves favorably when compared against the best evaluated GA algorithm designed for the same purpose reported

Table 5. Comparison between MDQL and MOMGA.

	MDQL		MOMGA	
	ϵ_{ratio}	σ	ϵ_{ratio}	σ
MOP1	30-45	0.00-0.04	NA	0.00-0.50
MOP2	65-70	0.02-0.09	92-100	0.02-0.17
MOP3	60-75	0.00-0.80	NA	0.00-1.07
MOP4	10-23	0.00-0.05	20-60	0.00-0.03

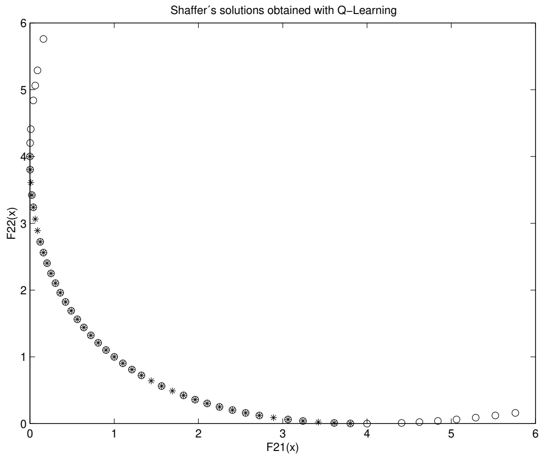

Figure 5. True vs calculated (known) Pareto front for MOP1 obtained with MDQL, * represent PF_{true}, while $\circ$ represent PF_{MDQL}.

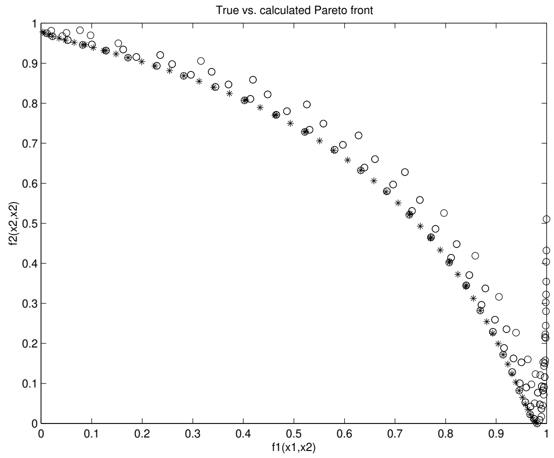

Figure 6. True vs calculated (known) Pareto front for MOP2, * represents PF_{true}, while $\circ$ represents PF_{MDQL}.

in recent literature (Veldhuizen, 1999). In contrast to other algorithms, no special attention was given to the fine tuning of parameters, as a uniform criterion was used for all test cases.

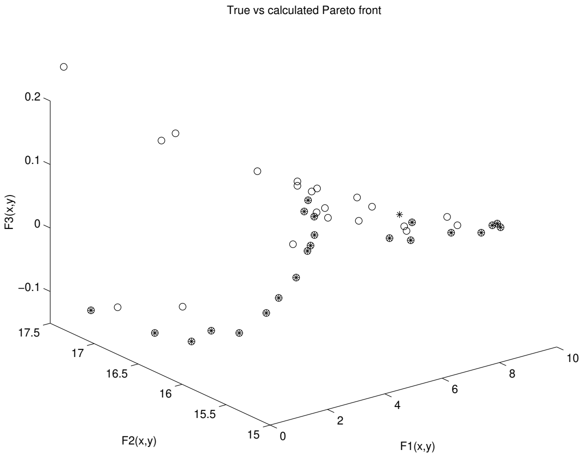

Figure 7. True vs calculated pareto front for tri-objective MOP3, * represent PF_{true}, ∘ represent PF_{MDQL}.

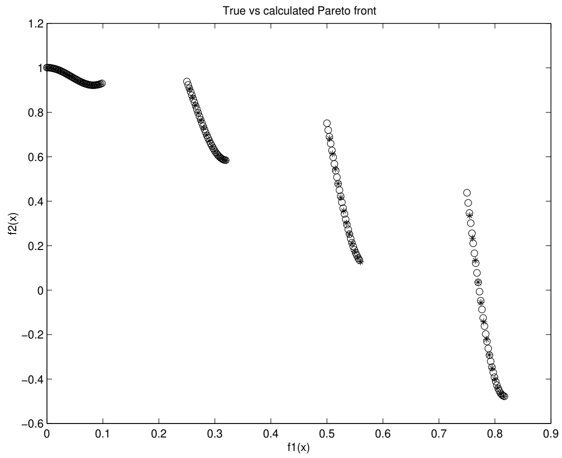

Figure 8. True vs calculated (known) Pareto front for MOP3, * represent PF_{true}, ∘ represent PF_{MDQL}.

5. Conclusions

This paper introduced a new distributed reinforcement learning algorithm, called DQL, that uses "traces" of all the other agents, generated from transitions between states, to infer a policy based on the most frequently selected actions. It was shown how DQL can substantially reduce the number of iterations of a standard Q-learning algorithm. This paper also presented a new algorithm, called MDQL, for solving multiple objective optimization problems. Its main characteristics are: (i) a family of agents for each objective, (ii) a negotiation mechanism to find compromise solutions, (iii) the delay of rewards given to the state-action pairs, and (iv) a dynamic adjustment of these rewards to guide the agents towards the solution of optimization problems. It was shown how MDQL was able to solve a wide range of multiple objective optimization problems in a competitive way when compare against a recent GA algorithm.

There are several future research directions that need to be addressed. In particular, we would like to measure how: (i) different policies, (ii) different values of the parameters involved and, (iii) a different negotiation mechanism, affect the performance of the algorithm.

References

Bilchev, George and Parmee, Ian. (1995) The Ant Colony Methaphor for Searching Continuous Design Spaces, in T. Fogarty, ed., *Lecture Notes in Computer Science 993*, Springer-Verlang, 1995, pp. 25-39.

Boutilier, Craig (1999). Sequential Optimality and Coordination in Multiagent Systems. *Proc. IJCAI-99*, Stockholm Sweden, July 31 - August 6, 1999

Claus, Caroline and Boutilier, Craig (1997). The Dynamics of Reinforcement Learning in Cooperative Multiagent Systems. *AAAI-97 Work. Multiagent Learning,* pp. 13-18, Providence, 1997.

Coello, Carlos. (1999), Comprehensive Survey of Evolutionary-Based Multiobjective Optimization Techniques. *Knowledge and Information Systems. An International Journal*, 1(3):269-308, August 1999.

Deb, Kalyanmoy. (1998), *Multiobjective Genetic Algorithms: Problem Difficulties and Construction of Test*

Problems. Technical Report TR CI-49/98, University of Dortmund, Germany: Department of Computer Science/XI, 1998.

Fonseca, Carlos M. and Flemming, Peter J. (1995a) An overview of evolutionary algorithms in multiobjective optimization. *Evolutionary Computation,* 3(1):1-16, 1995.

Fonseca, Carlos M. and Flemming Peter J. (1995b) "Multiobjective Genetic Algorithms Made Easy: Selection, Sharing, and mating Restriction." *Proceedings of the 1st International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications*.pp. 45-52. September: IEEE, 1995.

Gambardella, L.M., Dorigo, M. (1995) Ant-Q: A Reinforcement Learning approach to the traveling salesman problem. In *Proceedings of the 12th. International Conference on Machine Learning* (ML-95), pp., 252-260, Morgan Kaufmann.

Hu, Junglin and Wellman, Michael P. (1998). Multiagent Reinforcement Learning: Theoretical Framework and an Algorithm. *Proc. of the Fifteenth International Conference on Machine Learning*, pp. 242-243.

Littman, Michael (1994). Markov Games as a Framework for Multi-agent Reinforcement Learning, *Proc. of the Eleventh International Conference on Machine Learning*, pp. 157-163, New Brunswick, NJ, 1994.

Mariano, C., Morales E. (1999a). MOAQ an Ant-Q Algorithm for Multiple Objective Optimization Problems. In Banzhaf, W., Daida, J., Eiben, A. E., Garzon, M. H., Honavar, V., Jakiela, M., & Smith, R. E. (eds.). *GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference* (pp. 894-901). July 13-17, 1999, Orlando, Florida USA. San Francisco, CA: Morgan Kaufmann.

Mariano, C., Morales E. (1999b). MOAQ a Distributed Reinforcement Learning Algorithm for the Solution of Multiple Objectives Optimization Problems. in *Memorias del Segundo Encuentro Nacional de Computación ENC99*, paper No. 111 . 12-15 Sept, Pachuca Hidalgo, Mexico, Sociedad Mexicana de Ciencias de la Computación.

Schott, J. (1995) *Fault Tolerant Design Using Simple and Multicriteria Genetic Algorithms.* MS thesis, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, Cambridge, Massachusetts, May 1995.

Schaffer, D. (1985). Multiple objective optimization with vector evaluated genetic algorithms. In J.J. Grefenstette (Ed.), *Genetic Algorithms and Their Applications: Proccedings of the First International Conference on Genetic Algorithms* (pp. 93-100). Hillsdale, NJ: Lawrence Erlbaum.

Sutton, R.S., Barto, A.G. (1998). *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, MA.

Tan, Ming (1993). Multi-agent Reinforcement Learning: Independent vs. Cooperative Agents, *Proc. of the Tenth International Conference on Machine Learning*, pp. 330-337, Amherst, MA,1993.

Valenzuela-Rendón, Manuel and Uresti-Charre, Eduardo. (1997) A non-generational genetic algorithm for multiobjective optimization. In *Proceedings of the Seventh International conference on Genetic Algorithms,* pp. 658-665, San Francisco, California, 1997, Morgan Kaufmann.

Van Veldhuizen, David and Lamont, Gary. (1998) Evolutionary Computation and Convergence to a Pareto Front, *Late Breaking Papers at the Genetic Programming 1998 Conference,* John R. Koza. pp. 221-228. Stanford, CA: Standford University, July, 1998.

Van Veldhuizen, David.(1999). *Evolutionary Algorithms: Classification, Analysis, and New Innovations.* PhD Thesis Dissertation, Faculty of the Graduate School of Engineering, Air Force Institute of Technology, Air University, June 1999

Van Veldhuizen, David and Lamont, Gary. (1999) Multiobjective Evolutionary algorithms Test Suites, *Proceedings of the 1999 ACM Symposium on Applied Computing,* Janice Carrol et al., editor pp. 351-357. 1999.

Viennet, R., et al., (1996), Multicriteria Optimization Using a Genetic Algorithm for Determining a Pareto Set, *International Journal of Systems Science,* 27(2):255-260 (1996).

C.J.C.H. Watkins.(1989) *Learning from Delayed Rewards.* Ph.D. thesis, Cambridge University. 1989.

Deniz Yuret. (1994) From Genetic Algorithms to Efficient Optimization, *Msc Thesis in Electrical Engineering and Computer Science,* Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, May 1994.

Eckart Zitzler and Lothar Thiele. (1998) Multiobjective Optimization Using Evolutionary Algorithms - A Comparative Case Study. In *Proceedings of the Parallel problem Solving from Nature- PPSN V* pp. 292-301. Amsterdam The Netherlands, September 1998, Springer