MOAQ a Distributed Reinforcement Learning Algorithm for the Solution of Multiple Objectives Optimization Problems.

C. E. Mariano
Instituto Mexicano de Tecnología del Agua
Paseo Cuauhnahuac 8532
Jiutepec, Morelos, 62550, MEXICO
cmariano@tlaloc.imta.mx

E. Morales
ITESM - Campus Morelos
Paseo de la Reforma 182-A
Temixco, Morelos, 62580, MEXICO
emorales@campus.mor.itesm.mx

Abstract

In this paper we study the application of Reinforcement Learning to the solution of Multiple Objectives Optimization Problems designing a new algorithm called MOAQ. MOAQ considers a family of agents for each objective function in the problem description. MOAQ generates compromise solutions using a negotiation mechanism between agents from different families. Non-dominated solutions receive a reward and reinforcement is performed to all the components of the non-dominated solutions. MOAQ was applied to the solution of two problems whose Pareto front was previously known. We compare and contrast the solutions obtained by MOAQ with the solutions obtained with two recently developed genetic algorithms, demostrating the ability of MOAQ to find and maintain the Pareto frontier.

1 Introduction

After about a decade since the pioneering work by Schaffer [10; 11], a number of studies on multiple objective genetic algorithms (GAs) have been generated (e.g., [5; 7; 8; 12; 14]). Although GAs have shown good behaviors in the solution of multiple objective optimizations problems, encoding is often a difficult procedure which may involve the simplification of some problem constraints.

On the other hand, reinforcement learning, especially Ant-Q, has shown good results in combinatorial optimization problems [3; 4; 6]. In reinforcement learning [13], an agent learns values trying to obtain the maximum reward based on proven actions that gave high rewards, while not forgetting to explore new states where reward values are unknown. In order to increase the exploration capabilities and to reduce convergence times, more than one agent can be used in the solution of a single problem, transforming the basic algorithm to a distributed reinforcement learning algorithm (Ant-Q). In this approach many agents try to satisfy a common objective; each agent proposes a solution which is evaluated and rewarded depending on its goodness.

In this work a new algorithm for multiple objective optimization problems called MOAQ is presented. MOAQ was tested in the solution of two two-objective problems whose Pareto frontier was known, proving the algorithm's capability to find and maintain the Pareto frontier during simulation.

Section 2 describes in detail the Multiple Objective Ant-Q Algorithm. Section 3 shows the results on the problems solved by MOAQ and compares them with other approaches. Section 4 discusses the algorithm and its performance, and finally conclusions and future research directions are given in section 5.

2 Multiple Objective Ant-Q

In reinforcement learning, an autonomous agent learns an optimal policy $\pi : S \rightarrow A$, 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 maximum discounted cumulative reward that can be achieved starting from state s and applying action a as the first action. In other words, the value of Q is the reward received immediately upon executing action a from state s, plus the value (discounted by γ) of following the optimal policy. Therefore the optimal policy for a given state s must select the action whose Q function value is maximum $\pi^*(s) = \arg\max_a Q(s, a)$. The idea is to learn $\widehat{Q}$ values that approach Q. The update of the $\widehat{Q}$ values is made according to the following rule:

$$\widehat{Q}(s, a) \leftarrow r(s, a) + \gamma \max_{a'} \widehat{Q}(s', a')$$

where $\widehat{Q}(s, a)$ is the updated $\widehat{Q}$ value for state s and action a, $r(s, a)$ is the reward the agent receives when

Table 1: MOAQ Algorithm.

Given: (i) a list of n families (objectives)
and (ii) m agents for each family
Let N be the maximum number of iterations
Let $IN := 0$ (initialize iteration number)
Initialize all the Q values for all the families
Until only non-dominant solutions or $IN > N$
  Let $IN := IN + 1$
  Initialize parameters of family 1
  for $j = 1$ to m
    find a solution for objective 1
  for $i = 2$ to n
    Initialize parameters of family i
    for $j = 1$ to m
      Use the solution found with ant j
      in objective $i-1$ to constraint the
      solution for ant j (of objective i)
      Find a solution for objective i
  Evaluate the found solutions
  for $j = 1$ to m
    if solution j violates any constraint,
      Apply punishment to all its
      components (corresponding agents)
    else if solution j is non dominated
      Apply reward to all its components
      Introduce solution j into the
      Pareto set
      Remove all dominated solution from
      the Pareto set
    else if solution j is dominated
      Neither apply reward nor punishment

it executes an action a in state s, and $\widehat{Q}(s', a')$ is the maximum $\widehat{Q}$ value for the next algorithm step.

Ant-Q was proposed in [6]. The algorithm is in essence a reinforcement learning algorithm with few changes improving its exploratory capabilities and convergence time. The basic changes, which are also used in MOAQ for single objectives, are:

- Ant-Q considers m agents, rather than a single one, trying to find a solution.

- Ant-Q considers a domain-dependent heuristic value indicating how good is to go from a particular state (s) to another state (s'), $HE(s, s')$.

- Ant-Q considers an action choice rule or state transition rule to select the next state for each agent using a combination of the heuristic value $HE(s, s')$ and the $Q(s, a)$ value associated with state s and the action a that takes the agent to state s'. Let us call it $COMB(s, s')$. The transition rule has the following form:

$$s = \begin{cases} \arg\max\ COMB(s, s') & \text{if } q \leq q_0 \\ P & \text{otherwise} \end{cases} \quad (1)$$

  where q is a value chosen randomly with uniform probability over $[0, 1]$, q_0 ($0 \leq q_0 \leq 1$) is a parameter such that the higher q_0 the smaller the probability to make a random choice, and P is a random state selected according to a probability distribution given as a function of the heuristic and the Q values.

- The delayed reinforcement is computed using the Q-learning rule [16]:

$$\widehat{Q}(s, a) \leftarrow (1 - \alpha) \cdot \widehat{Q}(s, a) + \alpha \left[r(s, a) + \gamma \max_{a'} \widehat{Q}(s', a') \right] \quad (2)$$

  where $\widehat{Q}(s, a)$ is the new $\widehat{Q}$ value for the s, a pair, α is the learning step, s' and a' are the state and action in the next algorithm step, and γ is a discount factor. The learned action-value function, $\widehat{Q}$, directly approximates Q, the optimal action-value function, independent of the policy being followed. All that is required for correct convergence is that all pairs are visited and updated. The computation of the delayed reward depends on the problem, so details of its computation will be given later in the paper.

In one iteration of the algorithm every agent proposes a solution, all the solutions are evaluated and the best solution receives a reward. The new $\widehat{Q}$ values are then reinforced using equation 2.

The basic idea behind MOAQ is to perform a similar optimization algorithm but with a family of agents for each objective. Each family tries to optimize an objective considering the solutions found for the other objectives. So that all the agents from the different families act in the same environment proposing actions and expecting a reward value which depends on how their actions helped to find trade-off solutions between the rest of the agents.

In our algorithm, all the families must have the same number of agents, say m. The order given to the objectives (families) is arbitrary, although some problems may need a particular order in which the objectives should be tried. As in Ant-Q, all the agents in one family try to find a solution at the same time. The m solutions found for one objective in one cycle influence the m successive starting points of the following family (objective). This process continues with all the families (objectives). Once all the families have been tried, a reward is given to the non-dominated solutions (to all the agents in each family responsible for that solution) that satisfy all the constraints. Solutions violating constraints are punished

while the rest of the solutions are neither rewarded nor punished. The whole process is repeated several times until only non dominated solutions (Pareto set) satisfying all the constraints are found or a predetermined number of iterations is satisfied (see table 1).

3 Application to two problems

MOAQ has been applied to other multiple objective optimization problems [9], however, in this paper we compare MOAQ against two artificial problems involving two objectives previously solved with genetic algorithms [7; 8; 14].

In these problems, MOAQ needs to find a binary string, where each agent has to decide whether to place a one or a zero in each bit position. The application of MOAQ to binary strings suggests that many problems that have previously been treated with genetic algorithms are susceptible to be solved with MOAQ.

3.1 Problem 1. A simple test function

We begin by testing MOAQ to a simple artificial problem proposed in [7] whose Pareto optimal front can be easily evaluated. The problem involves two objectives which are simple functions of finite bit string length l. Unitation, $Unit[s]$, is the number of ones in the string s. Pairs, $Prs[s]$, is the number of pairs of adjacent complementary bits, either 01 or 10. The objective is to maximize both functions.

For the solution of this problem each agent in both families has an associated string named $Tour$ of length l where it stores its solution and two variables named one_a^f and $pair_a^f$, where agent a of family f stores the number of ones and pairs, respectively, in its solution.

A solution is constructed by placing every agent from $family1$ on the left-most position of the string. At this point their corresponding one_a^f and $pair_a^f$ for all the agents are initialized to zero. Each agent decides, according to the transition rule in equation 3 (see below) whether a 0 or a 1 should be placed in its current string position, while the agent moves to the next string position.

$$b' = \begin{cases} \arg\max_a [HE(b,a) \cdot Q(b,a)] & \text{if } q \leq q_0 \\ bit & \text{otherwise} \end{cases} \quad (3)$$

where $HE(b,a)$ is the heuristic value for agent k in $family1$ used to evaluate how good is to select, in string position b, any of the two options (0 or 1) taking the agent to the next string position. $HE(b,a) = one_k^1$, which means that it is incremented if the action chooses a 1 and remains the same otherwise.

$$HE(b,a) = \begin{cases} HE(b,a)+1 & \text{if selection is 1} \\ HE(b,a) & \text{if selection is 0} \end{cases}$$

Table 2: String problems parameter setting.

Problem	q_0	α	γ	*Agents*	$Time^a$
12 bit	0.7	0.1	0.3	100	288 s
28 bit	0.7	0.1	0.3	250	358 s

[a] MOAQ is written in Borland C++ V4.5 running on a 75 Mhz Pentium PC.

The Q values are initialized to 1, q and q_0 are explained in section 2 (equation 1), *bit* is a random selection according to a uniform probability distribution:

$$\frac{HE(b,a) \cdot Q(b,a)}{\sum_{action} [HE(b,action) \cdot Q(b,action)]} \quad (4)$$

Once all agents in $family1$ have completed their strings, solutions are given to the corresponding agent in $family2$. Every agent in $family2$ maximizes $prs[s]$ following a process similar to that described for agents in $family1$. In this case, their heuristic is equal to the number of pairs, i.e., $HE(b,a) = pair_k^2$. The heuristic value increases if the number of pairs for $agent_k^2$ increases and does not produce more ones than those produced by $agent_k^1$. The intention is to relocate the ones proposed by $agent_k^1$ to positions where they contribute to maximize $prs[s]$. The heuristic value for agents in $family2$ is:

$$HE(b,a) = \begin{cases} HE(b,a)+1 & \text{if } pair_k^2 \text{ grows} \wedge \\ & one_k^2 \leq one_k^1 \\ HE(b,a) & \text{otherwise} \end{cases}$$

When all the agents from $family2$ have completed their strings an iteration has been performed and the solutions are evaluated. Non-Dominated solutions are rewarded with $r(b,a)_k = 1$, and their Q values are updated.

Figure 1 shows the results obtained after 20 algorithm iterations, where the numbers indicate how many occurrences of that solution were found. We can see that MOAQ succeeds in finding all but one member of the Pareto set, maintaining the Pareto set while searching the solutions' space. We have plotted the solutions over many iterations noting that the algorithm maintains similar number of solutions for each Pareto point, as shown for 100 iterations in Figure 2.

We have observed similar behavior on larger problems, with corresponding larger families. Figure 3 and 4 show the agents behavior after 20 and 100 iterations for the 28 bit string case. It can be seen that the algorithm is also capable in this case to find and maintain the Pareto front. The values of the parameters used for both cases are showed in Table 2.

MOAQ solutions for this problem solution are better than the Niched Pareto Genetic Algorithm results reported in [7; 8], since the spread and number of solutions

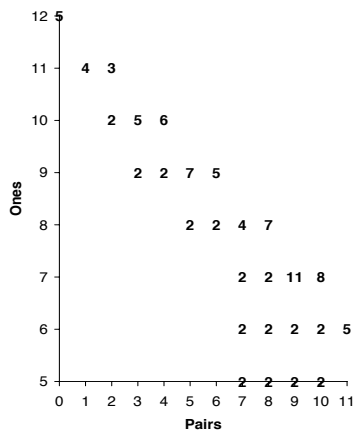

Figure 1: Solutions found after 20 iterations with a 12 bit string.

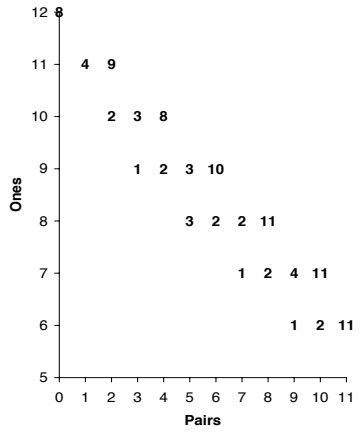

Figure 2: Solutions found after 100 iterations with a 12 bit string.

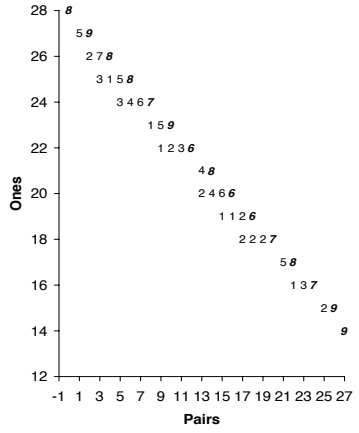

Figure 3: Solutions found after 20 iterations with 28 bit string.

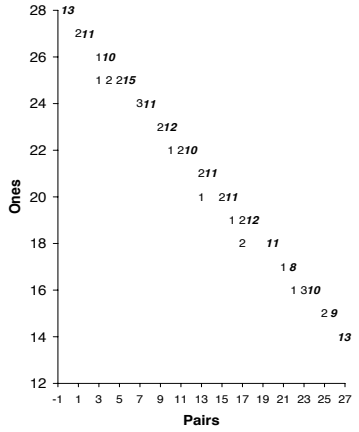

Figure 4: Solutions found after 100 iterations with 28 bit string.

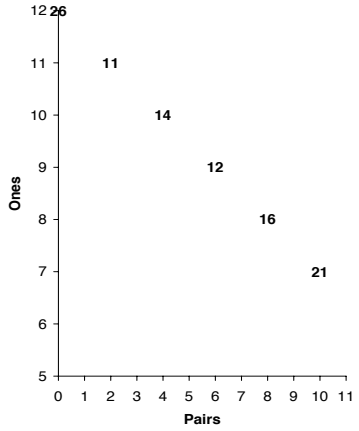

Figure 5: Horn's solutions for 12 bit at generation 100.

across the Pareto set are better, furthermore, MOAQ is able to find all the solutions in the Pareto front while the Niched Genetic Algorithm fails finding solutions in the left side of the front, [11,6] and [27,14] for 12 and 28 bit respectively (see figures 5 and 6).

3.2 Problem 2: Schaffer's F2

Next we tested our algorithm on Schaffer's function F2, with a single decision variable, the real valued x, and two objectives, f_{21} and f_{22} to be minimized.

$$f_{21}(x) = x^2 \qquad f_{22}(x) = (x-2)^2$$

The solution to this problem has been reported in [7; 8; 11; 14] among others, where genetic algorithms need to perform Niches or use Non generational behavior to maintain solutions distributed over the Pareto front. We used the same representation used in these works where the decision variable is mapped to a 14 bit string as a binary-coded integer.

The string position value is decided considering only

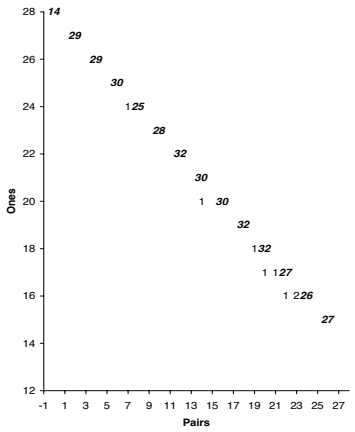

Figure 6: Horn's solutions for 28 bits at generation 200.

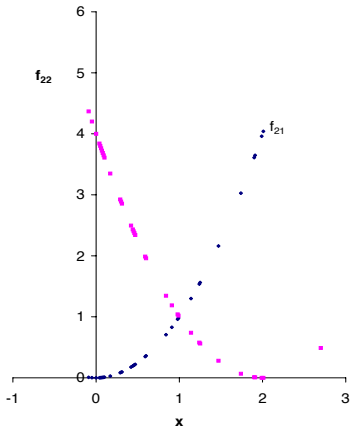

Figure 7: Expanded view of MOAQ's performance.

the Q values associated to every state-action pair $Q(b,a)$ using equation 5. The Q values are initialized to 1.

$$b' = \begin{cases} \arg\max_a [Q(b,a)] & \text{if } q \leq q_0 \\ bit & \text{otherwise} \end{cases} \quad (5)$$

In this case the stochastic selection (bit) is performed considering the following uniform probability distribution:

$$\frac{Q(b,a)}{\sum_{action} [Q(b,action)]} \quad (6)$$

Table 3 shows the parameter values used in the problem solution. In this case, a *greedier* selection was used to avoid a disperse Pareto set. More exploration (and diversity) can be obtained when q_0 is closer to 0.

Figure 8 shows the Pareto set obtained with MOAQ for Schaffer F2 problem after 100 iterations. Notice that almost all solutions lie on the Pareto front and some of them are very close to it. Figure 7 shows MOAQ's performance, and as can be seen, the solutions cover all

Table 3: Parameter setting for Shaffer F2.

Problem	q_0	α	γ	*Agents*	*Time*
F2	0.9	0.3	0.5	200	658 s

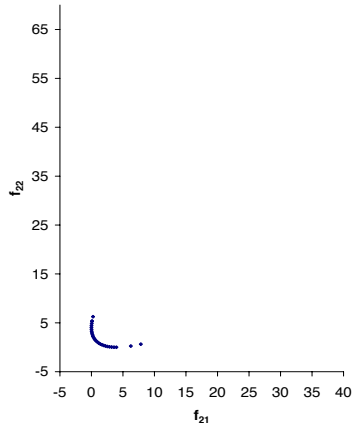

Figure 8: Pareto front for Schaffer's function F2.

the Pareto set. There are very few dominated solutions due to the exploration mechanism of the algorithm. The algorithm maintains the Pareto front over time and does not converge to a single solution, as often happens with genetic algorithm approaches, unless niching, crowding or some other similar technique is used.

Results reported by [7; 8] show disperse solutions in the Pareto front, in contrast with the results reported in [14] with a Non Generational Genetic Algorithm, which are similar to the solutions obtained with MOAQ.

4 Discussion

From the results obtained in the previous section, we can conclude that MOAQ is adequate to solve some multiple objective optimization problems. In particular, the encoding used suggests than many problems that have been treated with genetic algorithms can be solved with MOAQ. Furthermore, at least in these two problems, MOAQ showed comparatively better results than those obtained with genetic algorithms, although a more thorough comparison is needed.

MOAQ has been applied to the design of water distribution irrigation networks[9]. In this case the encoding of the problem is made in terms of the mathematical constraints involved in the problem, without having to transform the problem into an adequate binary code. In this aspect MOAQ presents a clear advantage over genetic algorithms approaches because MOAQ does not strictly need to codify the problem into a particular representation (i.e., binary), being MOAQ's encoding much more transparent and expressive because it is made in terms of the mathematical constraints of the problem; avoiding the large proportion of the time often devoted,

when working with genetic algorithms, to find an adequate encoding of the problem.

Theoretical analysis of the algorithm is still required in terms of stability, but as can be seen from the results, MOAQ looks as a promising alternative for the solution of multiple objective optimization problems.

A careful sensibility analysis is required for the parameters involved in the algorithm (α, γ). In this work some variations of these parameters were tested and the values reported are those that gave us better results.

5 Conclusions and Future Work

In this paper, a new algorithm called MOAQ for the solution of multiple objective optimization problems has been presented. MOAQ uses a family of agents for each objective and by sharing partial results between agents, global compromise solutions can be found. In particular, MOAQ was used to solve successfully some simple multiple objective optimization problems that have been solved with genetic algorithms with comparatively better results.

We would also like to test MOAQ on other real multiple objective optimization problems, especially those presented in engineering design and problems which are considered difficult for genetic algorithms (e.g., [1; 2; 15]).

References

[1] Deb, K. (1998). Multi-Objective Genetic Algorithms: Problem Difficulties and Construction of Test Problems, *Technical Report CI-49/98*, Dortmund: Department of Computer Science/LS11, University of Dortmund, Germany.

[2] Deb, K. (1999). Evolutionary Algorithms for Multi-Criterion Optimization in Engineering Design. In *Proceedings of Evolutionary Algorithms in Engineering and Computer Science* (EUROGEN'99).

[3] Dorigo, M., Maniezzo, V., Colorni, A. (1996). The Ant-System: Optimization by a colony of cooperating agents, *IEEE Transactions on Systems, Man and Cybernetics*, 26(1): 1-13 (Part B).

[4] Dorigo, M., Gambardella, L.M. (1997). Ant Colony System: Cooperative Learning Approach to the Traveling Salesman Problem, *IEEE Transactions on Evolutionary Computation*,1(1): 53-65, April 1997.

[5] Fonseca,C.M., Flemming, P.J. (1993). Genetic algorithms for multi-objective optimization: Formulation, discussion and generalization. In S. Forrest (Ed.), Genetic Algorithms: *Proceedings of the Fifth International Conference* (pp. 416-423). San Mateo, CA: Morgan Kaufmann.

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

[7] Horn J., Naftopolis, N. (1993) Multiobjective optimization using the niched Pareto genetic algorithm. IlliGAL Report 93005, University of Illinois at Urbana-Champaign.

[8] Horn J., Naftopolis, N., Golberg, D.E. (1994). A niched Pareto genetic algorithm for multi-objective optimization. In *Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence*, vol. 1 (pp. 82-87). New York: IEEE.

[9] Mariano, C., Morales E. (1999). 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.

[10] Schaffer, D., (1984). Some Experiments in machine learning using vector evaluated genetic algorithms (Doctoral Dissertation). Nashville, TN: Vanderbilt University.

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

[12] Srinivas, N., Deb, K. (1994). Multiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation*, 2(3), 221-248.

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

[14] Valenzuela-Rendón, M., Y Uresti-Charre, E. (1997). A non generational genetic algorithm for multiobjective optimization. In *Proceedings of the seventh international conference on genetic algorithms*, pp. 658-665.

[15] Van Veldhuizen, David A. *Multiobjective Evolutionary Algorithms: Clasifications, Analises and New Innovations*. PhD Thesis, AFIT/DS/ENG/99-01, Air Force Institute of Technology, Wright-Patterson AFB, June 1999.

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