Incremental Refinement of Solutions for Dynamic Multi Objective Optimization Problems

Carlos E. Mariano-Romero[1] and Eduardo F. Morales M.[2]

[1] Mexican Institute of Water Technology, Paseo Cuauhnáhuac 8532
Jiutepec, Mor, 62550, Mexico,
[2] INAOE, Luis Enrique Erro No. 1
Santa Maria Tonantzintla, Puebla, 72840, Mexico,

Abstract. *MDQL* is an algorithm, based on reinforcement learning, for solving multiple objective optimization problems, that has been tested on several applications with promising results [6, 7]. *MDQL* discretizes the decision variables into a set of states, each associated with actions to move agents to contiguous states. A group of agents explore this state space and are able to find Pareto sets applying a distributed reinforcement learning algorithm. The precision of the Pareto solutions depends on the chosen granularity of the states. A finer granularity on the states creates more precise solutions but at the expense of a larger search space, and consequently the need for more computational resources. In this paper, a very important improvement is introduced into the original *MDQL* algorithm to incrementally refined the Pareto solutions. The new algorithm, called *IMDQL*, starts with a coarse granularity to find an initial Pareto set. A vicinity for each of the Pareto solutions in refined and a new Pareto set is founded in this refined state space. This process continues until there is no more improvement within a small threshold value. It is shown that *IMDQL* not only improves the solutions found by *MDQL*, but also converges faster.
MDQL has also been tested on the solutions of dynamic optimization problems. In this paper, it is also shown that the adaptation capabilities observed in *MDQL* can be improved with *IMDQL*. *IMDQL* was tested on the benchmark problems proposed by Jin [4]. Performance evaluation was made using the Collective Mean Fitness metric proposed by Morrison [10].
IMDQL was compared against an standard evolution strategy with the covariance matrix adaptation (CMA-ES) with very promising results.

1 Introduction

One of the most important reasons for the growing interest in the solution of dynamic optimization problems is that many real-world optimization problems are not stationary. To solve this type of problems the optimizer must be able to adapt itself during optimization to track the optimum. Some methods have

been proposed to solve not stationary optimization problems, most of them are based in some of the following strategies to increase the ability of the methods to track moving optima [4]:

- Maintain population diversity by inserting randomly generated individuals, Niching or reformulating the fitness function considering the age of individuals or the entropy of the population.
- Memorize the past using redundant coding, explicit memory or multiple populations.
- Adapt the strategy parameters of the evolutionary algorithms.

MDQL is a multiple objective optimization algorithm that uses a distributed reinforcement learning approach to find Pareto sets. It has been tested on several bechmarking problems and on real world highly constrained optimization problems [5] [7].

MDQL first discretizes the decision variables into a set of states,$\mathcal{S}$, each associated with a set of actions $\mathcal{A}$ and for each action a value function to move agents to contiguous states. A group of agents explore this state space or environment and use a distributed reinforcement learning algorithm to find Pareto sets. The precision of the Pareto solutions depends on the chosen granularity of the environment. A finer granularity creates more precise solutions but at the expense of a larger search space, and consequently the need for more computational resources.

In this paper an incremental refinement of the environment is proposed to find more precise Pareto sets without additional computational resources. A course environment is initially constructed and a Pareto set is found using *MDQL*. The Pareto solutions found in the current environment are then considered as starting points of a more refined environments constructed in the vicinity of each of the Pareto solutions. This process continues until reaching a termination criterion. With the added capability of searching for solutions from a starting Pareto set, *IMDQL* is also able to dynamically adjust its Pareto front with variations in the problem conditions.

The main motivation behind modifications presented in *IMDQL* was the improvement of performance. In *MDQL* granularity in parameter spaces were always with the same size and static, constant granularity. Computational effort required to aproximate Pareto fronts, in constant granularity decision spaces, was measured almost constant along the search space. The use of incremental refinement approach is intented to aminorate the effort required to explore search space to identify feasible regions, using coarse partitions, and to increment efforts in feasible regions identified in order to approximate solutions with higher precision. Incremental refinement is made arround Pareto solutions identified.

Presented results correspond to dynamic optimization problems, one of the reasons to test *IMDQL* performance in the solution of this type of problems is that they represent a challenge for most of the otimization methods which have been applied to its solution. Another motivation is that previous estudies made over real multi-objective applications indicate that their nature is dynamic.

So the validation of this new approach is made over benchmarking dynamic optimization models as a preamble to attack more complex dynamic optimization problems.

IMDQL was tested on the solution of benchmark problem generated with the method proposed in [4] that considers a jumping optimum that change after the algorith converges. Previous charateristic is difficult to solve with traditional approaches as stated in [4]. Obtained solutions were comparared against Evolution Strategies, which was the approach used by Jin [4] to evaluate the method proposed. Performance measurement for both techniques over the same problem was made using Total Mean Fitness (F_T), and Collective Fitness (F_C) metrics proposed in [10].

The improvements over *MDQL* can produce more accurate results until a controllable degree of precision without seriously affecting the computational resources and can be used to shift the Pareto set with changing conditions in the problem description, both of which represent a significant improvement over *MDQL*, and allow it to be applied to more demanding optimization problems.

2 Incremental Multi objective Distributed Q Learning

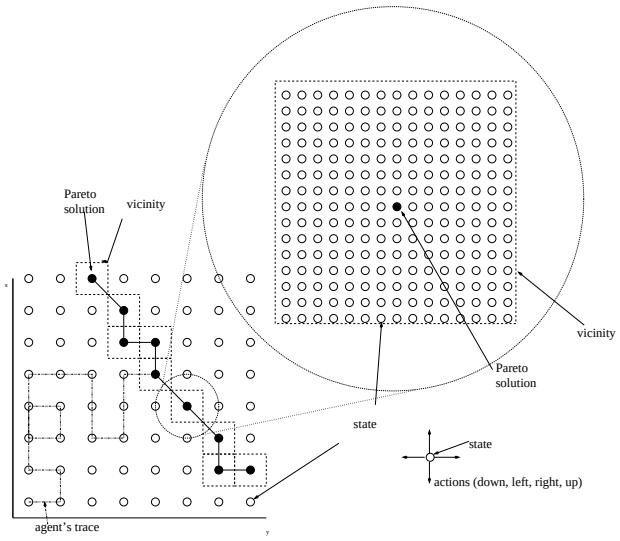

Fig. 1. *IMDQL*

IMDQL is based on *Q-learning* [12]. Its learned decision policy is determined by the state-action pair value function, $Q(s, a)$, which estimates long-term discounted rewards for each state-action pair. Given a current state $s \in \mathcal{S}$ and available actions $a_i \in \mathcal{A}_s$, a Q-learning agent selects most of the time an action

a with the highest estimated $Q(s,a)$ and with a small probability $\varepsilon \approx 0$, selects an alternative action. The agent then executes the action, receives an immediate reward r, and moves to the next state s'.

In each step, the agent updates $Q(s,a)$ by recursively discounting future utilities and weighting them by a positive learning rate α:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a' \in \mathcal{A}'_s} Q(s',a') - Q(s,a) \right] \tag{1}$$

where $(0 \leq \gamma \leq 1)$ is a discount parameter.

As an agent explores the state space, its estimate Q improves gradually, and, eventually, each $\max_{a' \in \mathcal{A}'_s} Q(s',a')$ approaches: $E\left\{\sum_{n=1}^{\infty} \gamma^{n-1} r_{t+n}\right\}$. Here r_t is the reward received at time t due the action chosen at time $t-1$. Watkins and Dayan [13] have shown that this Q-learning algorithm converges to an optimal decision policy for a finite Markov decision process.

Table 1. *IMDQL* algorithm.

First stage
(1) Initialize $Q(s,a)$ arbitrarily
Repeat (for n episodes)
  Initialize s, copy $Q(s,a)$ to $Q_C(s,a)$
  Repeat (for each step of episode)
    Repeat (for m agents)
      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 optimal Pareto solutions found and
  update the Q values in all Pareto solutions:
  $Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$
  Repeat (for m agents)
    Randomly select a solution
  Start a new episode
Second stage
Until termination criterion
  Construct a new environment for each solution
  solution and its vicinity
  Repeat (for p solutions)
    repeat (1)

IMDQL is presented in Table 1. In *IMDQL* all the agents have access to a temporary copy of the state-action pair evaluation functions ($Q_C(s,a)$). Each

time an agent has to select an action, it looks at this copy and decides, based on its information, which action to take. Once the agent performs the selected action, it updates the copy of the state-action value pair as follows, where $Q_C(s,a)$ represents a copy of the original $Q(s,a)$ pairs.

$$Q_C(s,a) \leftarrow Q_C(s,a) + \alpha \left[\gamma \max_{a' \in \mathcal{A}_s} Q_C(s',a') - Q_C(s,a) \right] \tag{2}$$

Updates are performed over copies of the original Q values and the original Q-values are consequently not affected at this stage. All the agents are moved one step at a time, updating and sharing their common Q_C values until reaching a stopping criterion. The agents use the copies of the Q values to decide which actions to take following an $\epsilon-greedy$ policy. The state values that are visited by the agents represent values for the decision variables of the multi-objective optimization problem. When all the agents have found a solution the Q value copies are discarded and the state-action pairs considered in the best solution receive a reward which reinforce their values according to Eq. 1. This updates the original Q-values from which a new copy is created for the next cycle. The whole process is repeated until reaching a termination criterion (see Table 1).

All the agents act on the same environment and have access to the same Q and Q_C values. The copies of the Q values are used as guidance to the agents of what seems to be promising states. However, only Pareto solutions found by all the agents receives an actual reward.

The previous procedure is repeated for a fixed number of episodes updating the current Pareto set with new Pareto solutions found after the completion of each episode. Solutions in the final Pareto set are used to construct new finer environments as illustrated in Figure 1. Finally, in the second stage the whole process is repeated for each of the environments constructed. Obtained Pareto sets for all environments are combined to construct the final Pareto set. Construction of new finer environments can be made until a desired level of accuracy is achieved.

2.1 Adaptation of *IMDQL* to dynamic optimization problems

In many real world applications, the values of the variables governing the problem can change over time, displacing the optimum and creating what is known as a non-stationary problem. The goal for this type of problems is to maintain an optimal condition in the face of varying conditions of the environment [11]. The search of the optimum then becomes a continuous process. According to the speed of movement of the optimum, it may be necessary to give the task to an automaton [11], but if the position of the optimum in a dynamic process is shifting very rapidly, the way in which the search process follows the extrema takes on a greater significance for the overall quality. In these cases iterative methods such as dynamic programming or stepwise optimization of Bellman are more adequate [11].

MDQL has strong affinity with the characteristics of dynamic programming, it satisfies Bellman's optimum principle and suffers the same deficiencies with

the dimensionality of problems [8]. *IMDQL* reduces these deficiencies by starting from previous Pareto sets fronts and using a variable abstraction level thus reducing the dimensionality problem.

The main consideration in the application of *IMDQL* is that agents in the algorithm start from the solutions previously obtained. That is, *IMDQL* starts with a deterministic environment constructed with fixed values for the value functions for the first ladscape; when convergence is reached and a solution is obtained, a new cycle is started, changing the landscape to the next variation programmed. Agents start searching (adapting solution) from the existing environments which correspond to the solutions obtained for the previous landscape.

Searching for new solutions, from the last solution, given the landscape, significantly reduces the convergence time.

3 Test Problem

3.1 Benchmark Problem

Three types of dynamic test problems have been proposed to test optimization methods, in the most typical the optimum moves deterministically or stochastically during optimization. Another type of problems are those that use the variation of constraints as a dynamic scheduling problem. In general most of the dynamic optimization problems can be divided in a) problems for which the optimum moves linearly in parameter space with time; b) prblems for which the optimum moves nonlinearly in parameter space with time; c) problems for which the optimum oscillates periodically among a given number of points in parameter space deterministically; and d) problems for which the optimum moves randomly in parameter space with time.

Test problem generator proposed by Jin [4] considers dynamic weigthed aggregation (DWA) approach used in multi-objective optimization to construct dynamic optimizatiuon problems. Dynamism is generated variating weigths. The method ha the capability to construct problems with the characteristics mentioned for the four types of problems listed in previous paragraph. The use of DWA is based on its capabilities to reach solutions in the concave regions of multi-objective problems, good distribution of solutions along the Pareto set, which have been previously analyzed in [3]. Also, using DWA it is possible to construct dynamic optimization test problems being an efficient, easy tunable and functionally powerfull tool to generate dynamic optimization test problems [4]. Other test functions for dynamic optimization problems were previously suggested from different authors [9], but these functions are designed for single objective optimization.

More critical for most evolutionary algorithms is to track a jumping optimum after algorithm has converged [1],[4] and [10]. Equation 3 represents a dynamic optimization problem when the weight change between 0.2 and 0.8 in every 50 generations.

$$F(x) = w \sum_{i=1}^{n} x_i^2 + (1.0 - w) \sum_{i=1}^{n} (x_i - 2)^2 + 1, \tag{3}$$

3.2 Performance metric

Traditional meatrics in dynamic environments have not the capability to measure adaptation capabilities of heuristics after a landscape change. Some researchers have suggested the use of there adapted performance metrics based on a) the evaluation of the distance between the optimum value and the value of the best individual in the environment just after the envinonment change; b) off-line measure, where the best-so-far is reset at each fitness landscape change; c) Euclidian distance to the optimum in each generation; d) best-of-generation averages, at each generation, for many runs; the best-of-generation minus the worst within a small window of recent generations, compared to the best within the window minus the worst within a window.

The first two measures require knowledge of the generation when the fitness ladscape changed. In many real problems, and in some test problems obtaining this information may be difficult. The third measure, the Euclidean distance to the optimum is only available in test problems where the global optimum is already known. The forth is the most commonly reported metric has not the capability to measure the perfromance across the entire range of landscape dynamics, just over especific generations. The fifth measure assumes that the best fitness value will not change much over a small number of generations and does not provide a convinient method for comaring the full range of landscape dynamics [10].

Since we are concerned with the evaluation of performance across the entire range of landscapes dynamics, the experimental unit is considered to be the entire fitness trajectory, collected across the heuristic exposure to a large sample of the landscape dynamics. In this sense Total Mean Fitness (TMF) is proposed in Equation 4.

$$T_F = \frac{\sum_{m=1}^{M} \left(\frac{\sum_{g=1}^{G} F_{BG}}{G} \right)}{M} \tag{4}$$

Where: F_T = Constant, for $G = \infty$ is the total averge fitness for the heuristic over its exposure to all the possible landscape dynamics; F_{BG} is the best-of-generation; M is the number of runs of the hueuristic; G is the number of generations.

As can be noted a very large experiments are required to use this performance metric. But it is possible that an heuristic approches a constat just after fewer generations, depending on the behavior of the heuristic during the solution of an especific problem. This situation could be presented if: a) the heuristic has a reasonably recovery time for all type of landscape changes; b) The global maximum fitness can be assumed to be restricted to a relatively small range

of values. In order to evaluate the satisfaction of any of this two conditions Collective Mean Fitness (F_C) metric is proposed in Equation 5.

$$T_C = \frac{\sum_{m=1}^{M} \left(\frac{\sum_{g=1}^{G\prime} F_{BG}}{G\prime} \right)}{M} \approx T_F \tag{5}$$

Collective Mean Fitness (CMF)is defined as the average best-of-generation values, averaged over a sufficient number of generations, $G\prime$, required to expose the heuristic to a representative sample of all possible landscape dynamics, further averaged over multiple runs. CMF will aproach TMF after a sufficiently large exposure to the landscape dynamics. The value for $G\prime$ has to be defined experimentally evaluating the fluctuations of F_C over a fixed number of generations.

4 Results

The tracking performance of *IMDQL* for $n = 3$ is presented in Figure 2. To solve the problem presented in Eq. 3. Parameter space was partioned with initial increments of x equal to 0.1. One step of refinement were considered to reach final incremetns for x equal to 0.01. *IMDQ* operation parameters used were: $\alpha = 0.1$, $\gamma = 0.9$ with an ϵ-greedy strategy with $\epsilon = 0.1$. Dashed line in 2 represent real solution for the test problem represented in 3, solid line represent the solutions obtained with *IMDQL*.

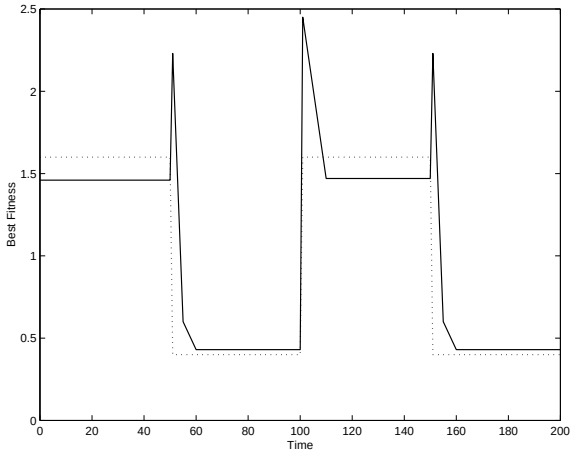

Fig. 2. Tracking a jumping optimum using *IMDQL*

As mentioned in previous paragraphs, benchmarking problem in Eq. 3 represents a dynamic optimization problem when the weight change between 0.2 and

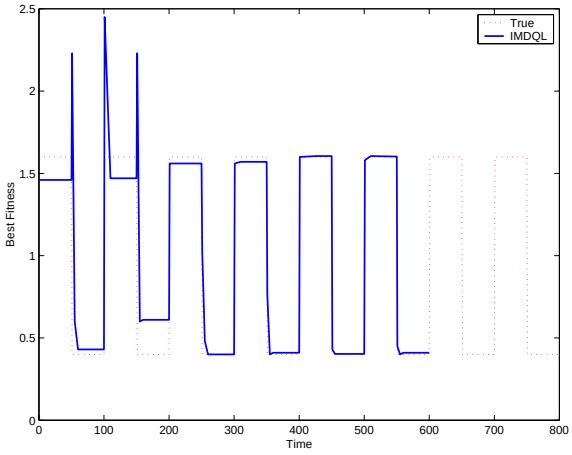

Fig. 3. Adaptation of *IMDQL* to landscape changes

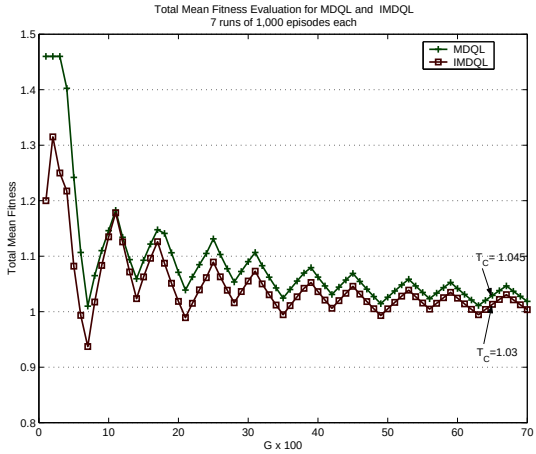

Fig. 4. Comparison between *IMDQL* and *MDQL* to landscape changes and evaluation of TMF metric

0.8 in every 50 generations. For *IMDQL* a generation is equivalent to an episode (see Section 2 for details). For these experiments *IMDQL* a run consist of $1,000$ episodes (200 landscape changes). It can be appreciated in 2, represented with solid lines the behavior of *IMDQL* to landscape changes during its execution. It can be also appreciated, represented with the dotted line, the exact solution to the benchmarking problem used.

Figure 3 present the behavior of *IMDQL* after the execution of several episodes. It can be appreciated that obtained solutions get closer to the real solutions. Another important fact to be noted is *IMDQL* estability of soltutions with time, that is, algorithm's capabilities to adapt when changes in lanscape are presented. This capability is measured using TMF and CMF metrics. Results from the evaluation with both metrics are presented in Figure 4.

In order to compare the performance of *IMDQL*, the CMA-ES obtained from [2] was executed considering a parent and offspring population sizes of 15 and 100 respecively and the initial stepsizes were configured to 0.1. No ellitism neither recombination were considered. CMA-ES is unable to track the moving optimum closely, the solution implemented with the CMA-ES was to check continuously the stepsizes, reducing its values to 0.1 when they grow-up. With this consideration TMF after 7 executions with $1,000$ generations each was equal to 1.183, compared with 1.0037 obtained with *IMDQL* it can be appreciated that adaptation capabilities of *IMDQL* are better.

5 Discussion

Considering these results it can be say that *IMDQL* can: (i) produce more accurate solutions, (ii) reduce convergence times, and (iii) adjust the Pareto set to changing conditions on the operation parameters of the problem.

6 Conclusions

References

1. J. Branke. *Evolutionary Optimization in Dynamic Environments*. Kluwer Academic Publisher, Boston, USA, 2002.
2. N. Hansen. CMA Evolution Strategy (Covariance Matrix Adaptation). http://www.bionik.tu-berlin.de/user/niko/cmaesintro.html.
3. Y. Jin, M. Olhofer, and B. Sendhoff. Evolutionary dynamic weighted aggregation for multiple optimization: Why does it work? In *Genetic and Evolutionary Computation Conference*, pages 1042–1049, San Francisco, CA, USA, 2001. IEEE.
4. Y. Jin and B. Sendhoff. Constructing Dynamic Optimization Test Problems Using the Multi-objective Optimization Concept. In G. R. R. et al., editor, *Applications of Evolutionary Computing. Proceedings of Evoworkshops 2004: EvoBIO, EvoCOMNET, EvoHOT, EvoIASP, EvoMUSART, and EvoSTOC*, pages 525–536, Coimbra, Portugal, April 2004. Springer. Lecture Notes in Computer Science Vol. 3005.

5. C. Mariano. *Reinforcement Learning in Multiobjective Optimization*. PhD thesis, ITESM Campus Cuernavaca, 2001.
6. C. Mariano, V. Alcocer, and E. Morales. Multi-objective optimization of water using systems. *European Journal on Operational Research*, 181(3):1691–1707, september 2007.
7. C. Mariano and E. Morales. A new approach for the solution of multiple objective optimization problems based on reinforcement learning. *Lecture Notes in Artificial Intelligence*, 1793(1):212–223, April 2000.
8. C. Mariano and E. Morales. A new updating strategy for reinforcement learning based on Q-learning. *Lecture Notes in Artificial Intelligence*, 2167(1):324–335, July 2001.
9. R. Morrison and K. D. Jong. A test problem generator for non stationary environments. In *Proceedings of Congress on Evolutionary Computation*, pages 2047–2053, 1999.
10. R. W. Morrison. Performance measurement in dynamic environmets. In *Proceedings of the Workshop on Evolutionary Algorithms for Dynamic Optimization Problems (GECCO 2003)*, 2003.
11. H. Schwefel. *Evolution and Optimum Seeking*. John Wiley and Sons, New York, USA, 1995.
12. C. Watkins. *Learning from Delayed Rewards*. PhD thesis, Cambridge University, 1978.
13. C. Watkins and P. Dayan. Q-learning. *Machine Learning*, 3(1):279–292, 1992.