A fuzzy-set based strategy in dynamic environments

Carlos Cruz Corona
Dep. Comp. Science and AI
University of Granada
carloscruz@decsai.ugr.es

David Pelta
Dep. Comp. Science and AI
University of Granada
dpelta@decsai.ugr.es

Jose Luis Verdegay
Dep. Comp. Science and AI
University of Granada
verdegay@decsai.ugr.es

Abstract

Many complex real-world optimization problems are dynamic. In order to approach them is necessary to have tools that are able to adapt to the changes that take place in the time. In this work we propose a strategy that jointly use a set of solutions and a set of simple agents. Implicit and explicit memory mechanisms are used and we analyze the behavior of the strategy when coupled with a fuzzy rule to control the updating of the solution's set. Tests are performed on the moving peaks benchmark problem under different scenarios.

Keywords: Cooperative Metaheuristics, Dynamic Optimization, Soft Computing.

1 Introduction

An intelligent system is a logical or physical system that perceives its environment and takes actions which maximizes its chances of success. It is flexible to changing environments and changing goals, it learns from experience, and it makes appropriate choices given perceptual limitations and finite computation.

Using some type of intelligent systems, like metaheuristics, to solve complex problems with a single and perfectly defined objective, is a field where the necessity of new techniques is debatable. Their use in these problems is sufficiently validated, having an abundant and easily accessible bibliography [7, 13].

On the other hand, it is assumed that the challenge that will be to confront in the next years, will be associate to the problems with several objectives, that can vary with the time and where it can have non probabilistic uncertainty in the variables values, or the objective function is not accurately known and it must be determined for example through a simulation or subjective valuations of experts. These problems are referred as name of Dynamic Optimization Problems [1].

A Dynamic Optimization Problem, *DOP*, can be formally defined as follow:

$$DOP = \left\{ \begin{array}{l} optimize \quad f(x,t) \\ s.t. \quad x \in F(t) \subseteq S, \ t \in T \end{array} \right\}$$

where

- $S \in R^n$, S is the search space.
- t is the time.
- $f : S \times T \rightarrow R$, is the objective function that assign to each possible solution ($x \in S$) a numerical value ($f(x;t) \in R$) at time t
- $F(t)$, is the set of feasible solutions $x \in F(t) \subseteq S$ at time t.

The goal of the methods for dealing with *DOP* is no longer to locate a stationary optimal solution, but to track their progression through the space and time as closely as possible. Therefore, these methods will have to be able to incorporate theoretical and practical tools that allow them to approach the uncertainty, failures tolerance and inherent noise, and is in this field where the techniques and models based on Soft Computing could help to develop capable methods to adapt, reshape, and repair themselves [8].

Evolutionary Algorithms (EAs) are widely used to solve these *DOP* [1] because it is easier to follow the changes if a *set* of solutions is available instead of just a single one. However, EAs need to specialize to be able to detect the changes and to respond them. A way is using memory schemes that work by storing useful information, either implicitly or explicitly, from the current scenario and reusing it later in a new one.

An implicit memory scheme [9, 12] is an algorithm that uses representations containing more information than necessary and basically has some memory where good (partial) solutions may be stored and reused later as necessary. An explicitly memory scheme [10,11] is an algorithm that uses an extra storage space with explicit rules for storing and retrieving information.

Departing from the strategy presented in [14], we propose here a simple information sharing scheme based on a fuzzy rule, that can be understood as a cooperation strategy among agents and as a dynamic resampling of the population of solutions. The proposal is tested on the moving-peaks problem [1], a classical *DOP* benchmark.

The contribution is organized as follows: Section 2 describes in detail the proposed strategy, the use of memory and introduce the cooperation scheme proposed. Then, the proposal is tested on the moving peaks benchmark and the experimental results are presented in Section 4. The paper concludes with a summary and some suggestions for future work.

2 Description of the Proposal

It is well known that better problem solving strategies may be obtained through cooperation among solvers. Also, cooperation allows to obtain improvement in robustness and quality of the solutions [5].

The strategy proposed in this paper is based on the jointly use of two populations and a centralized information repository [14, 15].

The first population is composed by a set of solutions arranged in a matrix or "world" M with certain dimensions. Each cell $M(i, j)$ contains:

- a solution to the problem at hand, refereed to as $M(i, j).V$;
- and the cost of this solution $M(i, j).C$.

There is no topological relation between the positions of the solutions in the matrix and their corresponding costs.

The second population is made by a set of "agents" $\mathcal{A} = \{a_1, a_2, \ldots, a_k\}$ and we assume that the world M is larger than the number of agents. Each agent is represented by a 4-tuple: $a_i = \{move?, (x, y), currentSol, cost\}$, where $move? \in \{True, False\}$ states if a_i is free to move or not, (x, y) indicates the position where it lies and *currentSol* is the current solution being manipulated by a_i. The objective value of *currentSol* is stored in the variable *cost*. We will refer to the each component of a particular agent a_i using the "dot" notation, so $a_i.cost$ will stand for the cost of the solution that a_i stores.

The last element is a centralized information repository (*CIR*). In its simplest form, it stores the cost and the best solution seen so far in the system. We will come back to *CIR* later.

The global strategy is described in Alg. 1. After a random initialization of solutions and agents positions, several processes are applied. On *Move Agents*, each agent free to move (in terms of its control variable) "jumps" to a random neighboring cell in certain radius.

Method *Change Solutions* is applied for every agent as follows: first step is to copy the cell solution into the agent: $a_i.currentSol := M(a_i.x, a_i.y).V$ and $a_i.cost := M(a_i.x, a_i.y).C$. Then, a_i selects randomly a variable $x_i \in currentSol$ and it is perturbed by this way: $x_i + N(0,1)$, where $N(0,1)$ is a random number generated from a normal distribution with mean 0 and variance 1. Finally, the modified $a_i.currentSol$ is evaluated and the cost stored into $a_i.cost$.

Now, this new solution $a_i.currentSol$ may improve $M(a_i.x, a_i.y).V$ or not and actions should be taken for each option. These actions are encapsulated within the *Update-and-Cooperate* stage, which is fully described in Alg. 2.

The procedure *UpdateMemory* will be described later. The other initial steps are clear: if the new solution obtained by the agent is better than the one stored in the cell, then the cell's solution and its cost are updated. If the new solution is the ever best seen, then we put it in the *CIR*.

Algorithm 1 Main Program

```
Initialize Matrix of Solutions randomly
Set Agents in the Matrix
while (!END) do
  Move Agents
  Change Solutions
  Evaluate Solutions
  Update-and-Cooperate
  if (conditions OK) then
    Modify Problem
    Evaluate Solutions
  end if
end while
```

Up to this point, we have a set of agents behaving as hill-climbers, while keeping record of the best solution ever seen.

The proposed strategy has a straight implementation under the NetLogo [16] software package. NetLogo is a multi-agent programming language and integrated modeling environment. It is particularly well suited for modeling complex systems developing over time.

Algorithm 2 Update-and-Cooperate Method *(maximizing)*

```
for each Agent $a_i$ do
  UpdateMemory($a_i.currentSol$,$a_i.cost$)
  if ($a_i.tmpSol$ is better than $M(a_i.x, a_i.y).V$) then
    $M(a_i.x, a_i.y).V := a_i.currentSol$
    $M(a_i.x, a_i.y).C := a_i.cost$
    if ($a_i.currentSol$ is better than the one stored in CIR) then
      update CIR with $a_i.currentSol$, $a_i.cost$
    end if
  else
    applyRule
  end if
end for
```

Under NetLogo's terminology, the cells of our matrix M translates into "patches" and the agents into "turtles". Patches and turtles may store any kind of information and communication among them is straightforward. The *CIR* component is modeled as a global variable and the same approach was taken for Mem.

The execution of agents/turtles is asynchronous, thus allowing the simulation of parallelism on a single processor computer.

2.1 Memory and Cooperation Mechanisms

Now, we need to define what happens when a solution is not improved and it is here where cooperation should be defined. In this work, we understand cooperation as a mechanism for information sharing among the entities of the system.

The population of solutions has two purposes: first, to serve as an implicit memory that is evolved through the action of the agents, and second, as a communication channel for them, because the value of a particular cell is the result of the accumulated changes made by the agents during the run. So, we have here an *implicit cooperation mechanism.*

We propose here an *explicit cooperation mechanism* that is based on a very simple idea: the best solution stored in *CIR* is copied into those cells whose corresponding solutions were not improved.

This mechanism is not always triggered. This "broadcasting" of the best solution is controlled by a fuzzy-set based rule indicated in Alg. 2 in the procedure *applyRule*

The idea of this rule is to change the not improved solution if it is "bad" with respect to a reference set of values. This reference set of values is, in fact, an explicit memory implemented as a fixed length vector Mem that is stored in *CIR*. Each time the procedure *UpdateMemory* is executed, the current agent's solution and its cost are sent to Mem which is updated in a first-in first-out manner when becomes full. The length of Mem is fixed on $2 \times k$ (being k the number of agents). In this way, the last two solutions values obtained by the agents are recorded.

So, the fuzzy rule analyzes the quality of the solution obtained by the agent ($a_i.cost$) in terms of the history of values stored in Mem:

IF `the quality of the solution obtained by` $agent_i$ `is` **low**
THEN $agent_i$ `gets the best solution from` *CIR* `and replace the one in its cell`

The label *low* is defined as follows:

$$\mu(x) = \begin{cases} 0.0 & \text{if } x > \beta \\ (\beta - x)/(\beta - \alpha) & \text{if } \alpha \le x \le \beta \\ 1.0 & \text{if } x < \alpha \end{cases}$$

where $x \in [0, 100]$ is the percentile rank of a cost in the samples stored in Mem. Also, $\alpha = 20$ and $\beta = 30$. The *best solution* is the one stored in *CIR*. In short, what the rule says is: if the reported values by an agent are among the worse in the memory, then replace the corresponding solution in the cell where the agent is. This is implemented as $M(a_i.x, a_i.y).V :=$ best solution from *CIR*, and $M(a_i.x, a_i.y).V :=$ best cost from *CIR*.

If the rule is executed, then the agent will not move in the next iteration (we set $a_i.move? := False$).

3 Moving Peaks Problem

In the context of dynamic environments one of the most widely used benchmark problem is the moving peaks problem [1]. The idea is to have an artificial multi-dimensional landscape consisting of several peaks, where the height, width and position of each peak is altered slightly every time a change in the environment occurs.

The cost function for n dimensions and m peaks, has the following form:

$$F(\overrightarrow{x}, t) = \max_{i=1,\dots m} \frac{H_i(t)}{1 + W_i(t) \sum_{j=1}^{n} (x_j - X_{ij}(t))^2}$$

where $\overrightarrow{x} \in \Re^n$ is a particular solution, $X \in \Re^{m \times n}$ is the set of m peaks locations (we need n values for locating a peak) and $H, W \in \Re^m$ are the height and width of each peak. The cost of a solution is how far it is from the closest peak.

The coordinates, the height H and the width W of each peak are randomly initialized. Then, every Δe evaluations the height and width of every peak are changed by adding a random Gaussian variable. The location of every peak is moved by a vector $\vec{v}$ of fixed length s in a random direction for $\alpha = 0$ or a direction depending on the previous direction for $\alpha > 0$. Thus α is a correlation coefficient allowing to control whether the changes exhibit a trend or not.

More formally, given $\sigma \in N(0, 1)$ a change can be described as:

$$\begin{aligned} H_i(t) &= H_i(t-1) + 7 \times \sigma \\ W_i(t) &= W_i(t-1) + 0.01 \times \sigma \\ \vec{X}_i(t) &= \vec{X}_i(t) + \vec{v} \end{aligned}$$

Many techniques are being applied to solve this kind of problems. The interested reader may refer to the following compilations [2, 4, 17] for current lines of research on the topic.

When comparing different methods in dynamic environments it is not useful to report just sim-

Table 1: Experiment 1: Problem Settings

Parameter	Setting
Number of peaks p	10
Number of dimensions d	10
Peak heights	$\in [30, 70]$
Peak widths	$\in [1, 12]$
Evals between changes Δe	500, 1000, 1500, 2000
Change severity s	1.0
Correlation coefficient α	1

ple values like the best solution achieved as in the static problems counterpart. One quite accepted measure is the off-line error [1] or accuracy, which is the running average of the difference between the optimum values and the best solution encountered so far at any time. This measure is always greater or equal to zero.

$$offline\ error = \frac{1}{T}\sum_{t=1}^{T}(optimum - bestSolution)$$

4 Computational Experiments

The main goal of the experiments is to assess the behavior of rule that may trigger the explicit cooperation mechanism proposed.

Firstly, we analyze how the performance of the system vary when different settings for the rule are used. Besides, we explore how the performance is affected when the problem change's frequency Δe is modified.

Secondly, we made a further analysis trying to gain knowledge about the dynamic behavior of the strategy.

In this contribution, the world has size $M^{15\times 15}$ and 15 agents are used. When an agent moves, it chooses a random direction and go forward a random number of steps between 1 and 3.

There are 100 changes for every run, so each run finishes when $100 \times \Delta e$ evaluations were done. We perform 30 repetitions for each setting.

4.1 Rule settings and Performance

In this experiment we analyze different settings for the rule under the problem's parameters summarized in Table 1.

The parameter that govern the triggering of the rule is a λ value representing a threshold for the membership values of Eq. 2.1. In fact, the rule will be triggered if $\mu(x) \geq \lambda$, being x a percentile rank associated with a particular fitness value.

Table 2: Average and std. deviation values for offline error (over 30 runs).

Δe	$Fuzzy(\lambda = 0,5)$	$Fuzzy(\lambda = 0,7)$
500	31,12 (1,58)	32,05 (1,75)
1000	19,84 (1,63)	20,96 (1,26)
1500	14,64 (1,29)	15,20 (1,22)
2000	11,02 (1,01)	11,74 (1,03)

Table 3: Minimum offline error values.

Δe	$Fuzzy(\lambda = 0,5)$	$Fuzzy(\lambda = 0,7)$
500	28,20	28,44
1000	16,20	17,08
1500	12,37	13,06
2000	9,15	9,82

The potential values are $\lambda = \{0.5, 0, 7\}$. The higher the value of λ, the worst the fitness should be in Mem to fire the rule.

Table 2 displays the average and standard deviation values with the fuzzy rule for $\lambda = 0, 5$ and $0, 7$. It can be seen a slightly advantage on the average offline error using $\lambda = 0, 5$ over $\lambda = 0, 7$. Although small, differences have statistical significance (U Mann-Whitney test). Also, as Δe increases, the offline average error improves.

The minimum offline error ever achieved are shown in Table 3. Again, the values are lower as Δe increases, and using $\lambda = 0, 5$ leads to better results than when $\lambda = 0, 7$ is used.

4.2 On the Dynamical Behavior of the Strategy

Although the average offline error taken at the end of the run is a good indicator of the strategy's performance, it does not shed any light regarding how the different settings lead to different runtime behavior.

So, we performed several runs to gather information about the dynamic behavior of the system. These runs were restricted to $\Delta e = 1000$ and 50 problem changes were done. Then, for each value of $\lambda = 0.5, 0.7$, we did 20 runs. For every run, we recorded the average offline error, number of failures and number of replacements done before each peak change, leading to 50 samples per run. The sequences of peak changes was fixed among the runs, but the initialization of solutions and agents locations was randomly generated.

Table 4: Experiment 2: Parameter settings of the problem

Parameter	Setting
Number of peaks p	10
Number of dimensions d	5
Peak heights	$\in [30, 70]$
Peak widths	$\in [1, 12]$
Evals between changes $\triangle e$	5000
Change severity s	1.0
Correlation coefficient α	0

The behavior for the fuzzy rule for the different settings are shown in Fig. 1. Plot is slightly zoomed in for visualization purposes so less than 50 points are shown. Each point represents the average over 20 runs.

We can see that both values for λ produced quite similar results but, in general, the use of $\lambda = 0,5$ allows a better tracking of the optimum line.

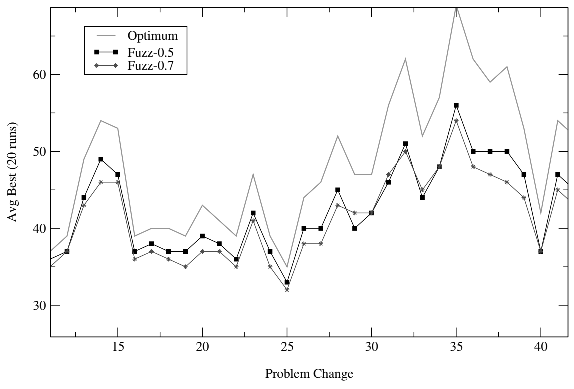

Figure 1: Average best fitness before each problem change for every setting.

4.3 Comparison with another algorithm

In this section we will compare our strategy against those recently published in [3], where the moving peak problem is defined with the settings summarized in Table 4. Such work may be considered as a state-of-art algorithm for dealing with the moving peak problem.

We should remark that measuring performance of algorithms on dynamic optimization problems is a hard task because of a lack of widely accepted criteria and standard formulations [6]. Moreover, although statistical testing is not possible because neither the data and the source code from Ref. [3] are available, it will serve us as a clear indication on the benefit of our proposal.

Table 5: Comparison between a multiswarm approach and our strategy

swarms	mCPSO	mQSO	agents	$\lambda = 0,5$
5(10+10)	3,74(0,14)	3,71(0,15)	5	1,57(0,07)
10(5+5)	2,05(0.07)	1,75(0,06)	10	1,87(0,15)
14(4+3)	2,29(0,07)	1,93(0,06)	14	2,05(0,18)
20(3+2)	2,89(0,07)	2,35(0.07)	20	2,31(0,22)
25(2+2)	3,27(0,08)	2,69(0,07)	25	2,47(0,23)

In Ref [3], Blackwell and Branke use a multiswarm approach composed of either neutral and charged particles or neutral and quantum particles. Several configurations are tested and they are defined as $M(N_1 + N_2)$, where M is the total number of swarms in the multiswarm, N_1 is the numbers of neutral particles and N_2 is the number of charged particles in the case of *mCPSO* (multi-Charged Particle Swarm Optimization) algorithm or quantum particles for *mQSO* (multi-Quantum Swarm Optimization) algorithm.

The results of both approaches appear in Table 5. The average offline error is clearly lower when our strategy use 5 agents, while a higher number of agents leads to similar or better results than certain configurations of mCPSO and mQSO.

In our strategy, as the number of agents increases, the offline error also increases. Recall that the problem change is made after certain number of evaluations. Within such period, we may assume we are dealing with a static problem. In this situation, the result implies that it is better to have a few agents doing several changes than more agents allowed to perform just a few modifications.

Looking at mCPSo and mQSO, it is not clear what may affect more the results: if the number of swarms available or their composition. In our case, we just deal with a single “entity” (agent) so we may claim, in principle, that our approach is simpler while obtaining competitive results.

5 Conclusions and Future Work

In this contribution we propose a scheme for updating the set of solutions in the context of a strategy that jointly use of a population of solutions and a set of simple optimizer agents.

The scheme is based on memory and a fuzzy rule that makes use of a history of previously seen costs to decide on the quality of solutions. This simple scheme, when used to update the pool of solutions, leaded to a successful optimization strategy.

The computational experiments clearly showed that the strategy obtained similar or better results than a multiswarm approach considered as a state-of-art algorithm for dealing with the moving peak problem.

Now, several lines of research are open. Among them, we wish to emphasize other alternatives for fusing Soft Computing' methodologies with computational strategies oriented to solve *DOP*. In this line and regarding *DOP*, the potential occurrence of uncertainties, dynamism and noise in the restrictions, parameters, cost function, etc, clearly recommends the use of concepts from the fuzzy sets and systems area to improve the modeling of the problem. Once this improved model is available, maybe new solving strategies could be needed, and here, the fields of multi-agent or cooperative heuristics, either centralized or descentralized, and governed by simple fuzzy rule bases, are going to be key elements in the quest for successful strategies for solving *DOP*.

Acknowledgements

This work was supported in part by Projects TIN-2005-08404-C04-01 (Co-HeuriFuzzy) and TIC-00129-JA (MINAS).

References

[1] J. Branke. *Evolutionary Optimization in Dynamic Enviroments*. Kluwer Academic Pub, 2002.

[2] J. Branke. Editorial: special issue on dynamic optimization problems. *Soft Computing*, 9(11), 2005.

[3] J. Branke and T. Blackwell. Multiswarms, exclusion, and anti-convergence in dynamic environments. *IEEE Transactions on Evolutionary Computation*, 10(4):459–472, 2006.

[4] J. Branke and Y. Jin. Special issue on evolutionary computation in the presence of uncertainty. *IEEE Transactions on Evolutionary Computation*, 10(4), 2006.

[5] T. Crainic, M. Gendreau, P. Hansen, and N. Mladenovic. Cooperative parallel variable neighborhood search for the p-median. *Journal of Heuristics*, 10:293–314, 2004.

[6] M. Dror and W. Powell. Stochastic and dynamic models in transportation. *Operations Research*, 41:11–14, 1993.

[7] F. Glover and G. A. Kochenberger, editors. *Handbook of Metaheuristics*. Kluwer Academic, 2003.

[8] N. Krasnogor, S. Gustafson, D. Pelta, and J. L. Verdegay, editors. *Systems Self-Assembly: multidisciplinary snapshots*. Elsevier, 2006.

[9] J. Lewis, E. Hart, and G. Ritchie. A comparison of dominance mechanisms and simple mutation on non-stationary problems. In A. E. Eiben, T. Bäck, M. Schoenauer, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature*, number 1498 in LNCS, pages 139–148. Springer, 1998.

[10] N. Mori, S. Imanishi, H. Kita, and Y. Nishikawa. Adaptation to changing environments by means of the memory based thermodynamical genetic algorithm. In T. Bäck, editor, *International Conference on Genetic Algorithms*, pages 299–306. Morgan Kaufmann, 1997.

[11] N. Mori, H. Kita, and Y. Nishikawa. Adaptation to a changing environment by means of the thermodynamical genetic algorithm. In H.-M. Voigt, editor, *Parallel Problem Solving from Nature*, number 1141 in LNCS, pages 513–522. Springer Verlag Berlin, 1996.

[12] K. P. Ng and K. C. Wong. A new diploid scheme and dominance change mechanism for non-stationary function optimization. In *Sixth International Conference on Genetic Algorithms*, pages 159–166. Morgan Kaufmann, 1995.

[13] P. Pardalos and M. Resende, editors. *Handbook of Applied Optimization*. Oxford University Press, 2002.

[14] D. Pelta and C. Cruz. A study on diversity and cooperation in a multi-agent strategy for dynamic optimization problems. *International Journal of Intelligent Systems*, 2008. In Press.

[15] D. Pelta, C. Cruz, A. Sancho-Royo, and J. Verdegay. Using memory and fuzzy rules in a co-operative multi-thread strategy for optimization. *Information Sciences*, 176:1849–1868, 2006.

[16] U. Wilensky. *Netlogo*. Center for connected learning and computer-based modelling., NorthWestern University, Evanston, IL, 1999.

[17] S. Yang. Associative memory scheme for genetic algorithms in dynamic environments. In F. Rothlauf, J. Branke, S. Cagnoni, and D. Corne, editors, *Applications of Evolutionary Computing*, volume 3907 of *LNCS*, pages 788–799. Springer-Verlag, 2006, 2006.