An Ant Colony Algorithm aimed at Dynamic Continuous Optimization

June 11, 2005

Abstract

The introduction of the concept of swarm intelligence into ant colony optimization (ACO) algorithms has shown the rich possibilities of self-organization when dealing with difficult optimization. Indeed, the inherent flexibility and efficiency of ACO algorithms proved to be advantageous for difficult dynamic discrete problems, e.g. routing in telecommunication networks. Moreover, we believe that ant colony algorithms can be efficient for both *continuous* dynamic problems and discrete ones. In order to exploit the features of these swarm intelligence algorithms for continuous dynamic optimization, we introduce an hybrid population-based ant colony algorithm. Considering the way ants communicate, we propose a "heterarchical" algorithm, called "Dynamic Hybrid Continuous Interacting Ant Colony" ($DHCIAC$), based on the hybridization of an "interacting ant colony" with a Nelder-Mead algorithm. Being confronted with the lack of benchmark functions for dynamic optimization in the literature, we have elaborated a complete set of various continuous dynamic problems. The efficiency of the proposed $DHCIAC$ algorithm is then demonstrated through numerous tests, conducted involving that new benchmark platform.

1 Introduction

In recent past, optimization of dynamic problems has evoked the interest of the researchers in various fields which has resulted in development of several increasingly powerful algorithms. Unlike in static optimization, where the final goal is to find a fixed global optimum, in dynamic optimization the aim is to find and follow the evolution of the global optimum during the entire optimization time.

Dynamic optimization has been recently studied under the name of "real-time optimization" (RTO) mainly in the field of model-based real-time optimization of chemical plants [15] [16] [17] and optimization of "non-stationary problems" [10] [3] or "dynamic environments" [5] [4] [12]. There are many fields that deal with dynamic optimization, our aim is to focus on the use of metaheuristics for

the optimization of difficult continuous dynamic problems. Only a few metaheuristics have been studied on such problems, mainly genetic algorithms [3], particle swarm optimization [12], etc. Ant colony algorithms have been employed on discrete dynamic problems [13, 6, 7] but not on continuous ones. Xiong and Jutan proposed a dynamic Nelder-Mead simplex method aimed at continuous optimization [15].

We propose in this paper a new algorithm inspired from the behaviour of ant colonies. The so-called "Dynamic Hybrid Continuous Interacting Ant Colony" (*DHCIAC*) algorithm is population-based, articulates itself around the notions of heterarchy and communication channel [9], and uses simplex algorithm as local search [8]. Two versions of this algorithm are proposed, using two different approaches for optimizing dynamic problems. The algorithm is tested on a newly proposed dynamic benchmark set, that attempts to cover a large scale of different dynamic difficulties.

This paper comprises of five more sections. In Sect. 2 we describe the continuous dynamic objective functions that we used to test the algorithm. In the subsequent Sect. 3 we present the *DHCIAC* algorithm. The setting of the parameters is studied in Sect. 4. Experimental results are then discussed in Sect. 5 and the conclusions make up the last section.

2 Test functions

The choice of the method for most effective comparison of algorithms for dynamic continuous optimization is quite difficult. Indeed we currently do not have, in dynamic environment, any set of benchmark problems which can be effectively employed to compare the performances of an optimization algorithm with those of its competitors. One of the objectives of this paper is to propose a set of dynamic test functions, that can be used as tools for comparison.

We can distinguish two main components of a dynamic optimization problem: the first one is the structure of the objective function, and the second one is the evolution of this structure. For the rest of this paper we will call the first component the *structure function*, and the second one the *time function*. Thus a dynamic objective function can be written as:

$$D(x,t) = S(x, T(t))$$

with S the structure function and T the time function. The relation between the structure function and the time function (i.e. *what* is undergoing changes) is called the "variation target".

2.1 Structure functions

The structure functions used are the classical analytical functions. In order to provide a set of coherent objective functions, we chose to determine which characteristics are the most important ones in the problems we encounter. We have determined twelve characteristics that can be used: continuity, convexity,

symmetry of the search space, symmetry of the function in its search space, quadratic nature (i.e. use of a polynomial function of the second degree in the problem), modality (i.e. presence of one or several optima), periodicity, number of local optima, number of dimensions, constraints, optimum position and shape of the basin of attraction.

2.2 Variation target

The variation targets can be classified in several categories:

- the variations of the whole function,
- the variations of the optimum only (only the optimum, and possibly its "bowl", are affected by the variation),
- the variations of the function but not of the optimum (the optimum is the only part which is not affected by the variation).

Each of these targets can change in position (the coordinates move along the search space) or in value (the value of the objective function is varying).

Moreover, there exist some more complex variations:

- the variation of the number of dimensions,
- the variation of the constraints,
- the phase shift (the target makes a rotation about one axis),
- and finally change in the whole structure (the objective functions drastically change in their structure function, for example from a convex to a concave shape).

2.3 Time functions

The time function can be linear (making the dynamic problem evolving infinitely), periodical (making the problem oscillate from a state to another) or more generally non-linear.

One important issue to be handled in the time function is the *relative speed*. Indeed, in real-time applications, we must consider the speed of the problem *in relation to the speed of the algorithm*. Thus we have three types of time functions: slow, fast and pseudo-discrete. A slow time function causes the dynamic problem to change a little between two calls (i.e. the model of the variations can be found). A fast time function causes the problem to change a lot between two calls, but the underlying model of the variations can still be found. In a pseudo-discrete time function, the variations are so fast that the problem looks completely different between two calls. In order to have an efficient way of determining when the relative speed is slow or fast, we use the notion of *optimization round*. The *length* of a round is defined as the number

of evaluations needed by an algorithm between two variations. For example, a round with a length of 10 will require ten calculations of the objective function. In our tests, a round of 20 evaluations is considered to be slow and a round of one evaluation is considered to be fast.

2.4 Benchmark set

We have used ten objective functions covering the different aspects of continuous dynamic optimization. The whole benchmark set used in this paper is presented in appendix A. The algorithms will be tested on each function, at three relative speeds, thus we have thirty different dynamic problems.

3 The *DHCIAC* algorithm

3.1 *HCIAC* algorithm

The *DHCIAC* algorithm is based on a static hybridized algorithm called *HCIAC* ("Hybrid Continuous Interacting Ant Colony" [8]) which is itself based on a static algorithm called *CIAC* ("Continuous Interacting Ant Colony" [9]). These ant colony algorithms are called heterarchical algorithms, and they use the concept of communication channels to gather informations about the problem to be optimized.

CIAC uses two communication channels, characterized by three properties (range, memory and type of information):

- The stigmergic channel calls upon the spots of pheromone, deposited within the search space, which are more or less attractive for the artificial ants, according to their concentrations and their distances. The characteristics of the stigmergic channel are the following ones: a) the "range" is maximal, *i.e.* all the ants can potentially take into account information, b) they utilize the concept of memory as the spots persist on the search space, and c) information evolves with time as the spots evaporate. The information carried by a spot implicitly contains the position of a point and explicitly comprises the value of the improvement found by the ant, having deposited the spot.

- The direct channel is implemented in the form of the messages exchanged between pairs of individuals. An artificial ant has a stack of received messages and can send some messages to another ant. Only one ant receives the message, the concept of memory is implemented in a messages stack that the ant memorizes, and finally information (here a couple position/value of a point) does not change with time.

In *HCIAC*, the decision between the deposit of a new spot or the reinforcement of an existing one is made according to a resolution parameter. In order to maintain the parallel design of the algorithm, each ant has its own resolution

parameter which we can call the "visible zone". The resolution parameter is dynamically set according to the ant's environment during the ant evolution within the search space. With this mechanism, the visible zone of the ant is automatically adjusted during the search process, according to the granularity of the search space it finds. From the point of view of the whole system, this is a resolution parameter as it describes the local granularity of the objective function and as it is decreasing during the search process. It should be noticed that the exponential decrease observed is not explicitely coded, but results from the whole system behaviour.

The main characteristic of *HCIAC* is its hybridization with the Nelder-Mead simplex algorithm. Many of the population-based algorithms are not very efficient for a fast and accurate determination of the local optima, but are quite good to locate promising areas. The ant colony algorithms perform better when they employ a local search, and for discrete problems, this technique is often used to make the ant algorithms competitive [2].

The Nelder-Mead algorithm [11] is a simple algorithm that presents the advantage of being a derivative-free method that can find the local optima fast. Instead of using the derivatives of the objective function, the Nelder-Mead algorithm uses a little population of points to handle a non degenerate "simplex". The simplex is a geometrical figure with a non-null volume in n dimensions that is a convex hull of $n+1$ dimensions.

The first step of the algorithm presented in Figure 1 is to randomly put the η ants over the search space according to an uniform distribution and to initialize all parameters.

Then the pheromonal spots evaporate, the value $\tau_{j,t+1}$ of each spot j at time $t+1$ is set according to:

$$\tau_{j,t+1} = \rho \cdot \tau_{j,t}$$

with ρ the persistence parameter.

At this step, a decision is taken, according to a threshold choice function: the ant chooses to handle one of the two communication channels. Here the two parameters of the stimulus-response function are called χ_τ for the threshold and χ_ρ for the power, these parameters are set for the whole population, as each ant i has a particular stimulus parameter χ_i, initialized at the first step according to a normal distribution $\aleph_{\chi_m,\chi_d}$.

If the ant chooses the stigmergic channel, it looks for pheromonal spots in the visible zone π_i which is initialized according to $\aleph_{\pi_m,\pi_d}$ at the first step. If there is some spots, it moves towards the weighted gravity center of the visible spots, else it goes on the step of motivation decision.

On the contrary, if the ant chooses the direct channel, it looks for messages in his message queue. If there is some messages, it moves towards the location indicated by the message and then adds some noise to its new location (i.e. it moves randomly in the visible zone), else it goes on the step of motivation decision.

If there is no spot and no message, the ant takes a decision based on its motivation ω_i. The choice is made according to a stimulus-response function,

the threshold ω_ρ and the power ω_τ are initialized at the first step for the whole population. The stimulus ω_i is initialized at 0 at the first step.

The first choice leads to the direct channel, with message handling. At this step, another decision is taken according to a stimulus-response function with the same parameter as the previous step, except that the stimulus is $(1 - \omega_i)$. If the choice is made to handle messages, a message is sent to a random ant and the motivation is increased of a small amount ω_δ. The second choice is to avoid messages, then the ant goes to the step of random move.

If the motivation choice is to handle local search, a Nelder-Mead search is launched with the location of the ant as base point and the radius of the visible zone as initial step length, the local search is limited to ν evaluations of the objective function. After this local search, the ant looks for visible spots. If there is a spot, it reinforces it according to:

$$\tau_{j,t+1} = \tau_{j,t} + \frac{\tau_{j,t}}{2}$$

and the radius of the visible zone is reduced to the distance to the farthest visible spot. Else if there is no spot, the ant deposits a new one, with a concentration set to the value of the objective function at this ant's current location. After the handling of the stigmergic channel, the motivation is set to zero.

3.2 *DHCIAC* algorithm

The *HCIAC* algorithm is well suited to dynamic problems due to its distributed and adaptive features. Indeed, *HCIAC* takes advantage of the heterarchical framework and makes use of communication channels, which can be used to rapidly diffuse an information about some variations in the objective function; besides *HCIAC* involves an efficient local search, which can be easily adapted to dynamic functions.

Two different versions of the *DHCIAC* algorithm have been developed so far: $DHCIAC_{find}$ and $DHCIAC_{track}$. Indeed, there exist two different strategies to find the global optimum when optimizing a dynamic problem [1]:

- find the optimum very quickly as soon as it changes,
- follow the evolution of the optimum.

DHCIAC can handle both strategies by tuning the local search part of the algorithm. The $DHCIAC_{find}$ version uses a classical Nelder-Mead search algorithm, whereas the $DHCIAC_{track}$ version uses a modified simplex algorithm, that does not reduce its size during its execution [15].

3.2.1 $DHCIAC_{find}$

The $DHCIAC_{find}$ demonstrates only a few differences with the *HCIAC* algorithm, as the "find" strategy is close to the one employed for static problems. The

modifications are aimed at changing the convergence behaviour and improving the speed of the algorithm.

Variations of the visible zone

The visible zone enables the *HCIAC* algorithm to focus its search behaviour on the promising zones of the search space. In *HCIAC*, when the visible zone of an ant changes, its size takes the value of the distance to the farthest spot in the visible zone. In $DHCIAC_{find}$, the size of the visible zone is set to the distance of the spot which permits the smallest change of this resolution parameter.

3.2.2 $DHCIAC_{track}$

The main difference with the *HCIAC* algorithm is that in $DHCIAC_{track}$ the local search algorithm is not the classical Nelder-Mead search but a newly proposed dynamic simplex [15].

Dynamic simplex

The dynamic simplex shows two main differences with the standard Nelder-Mead simplex algorithm. Firstly, it uses only the reflection operation, to keep a fixed size that permits to avoid the simplex to lock itself before the optimum can be tracked. Secondly, it only checks the best point to look for a variation. The complete algorithm is presented in algorithm 1.

Initial size of the simplex

One major parameter in the dynamic simplex algorithm is the size of the simplex, as it ideally should not change (as pointed out in [15]). As in *HCIAC*, in $DHCIAC_{track}$ each ant can launch a local search at any time, according to its "motivation state". Thus the size of the dynamic simplex is set according to the current size of the visible zone of the ant.

Variations of the visible zone

In $DHCIAC_{track}$ we use the same mechanism as in $DHCIAC_{find}$ to set the size of the visible zone.

4 Parameter setting

The setting of the parameters is a delicate problem, when different metaheuristics are employed. As illustrated by the concept of adaptive memory programming [14], there exist two main phenomena in metaheuristics: intensification and diversification. These guidelines are followed while setting the parameters of the two versions of the $DHCIAC$ algorithm. Indeed, $DHCIAC_{find}$ is aimed at searching new optima appearing during the optimization time, so that the correct parameter set can achieve a diversification behaviour. On the contrary, $DHCIAC_{track}$ tries to follow an evolving optimum, so that the better behaviour is an intensification one.

In the *HCIAC* algorithm, only two parameters have a significant impact: the maximum number of local search evaluations (ν) and the number of ants (η). We have found that it is possible to choose any satisfying compromise between an intensification and a diversification behaviour by tuning these two

parameters. In table 1 we present different values for ν and η according to an index P_i, which represents the balance between the greatest intensification behaviour ($P_i = 1$) and the greatest diversification behaviour ($P_i = 10$).

4.1 The *track* version

As $DHCIAC_{track}$ is built around an intensification idea, we have thus chosen to use the parameter set associated with $P_i = 3$: $\nu = 90$ and $\eta = 10$.

4.2 The *find* version

$DHCIAC_{find}$ makes use of a diversification behaviour, so that we have chosen to use the parameter set associated with $P_i = 7$: $\nu = 40$ and $\eta = 15$.

4.2.1 High evaporation rate

With the $DHCIAC_{track}$ version of the algorithm, it is necessary to avoid being trapped into non moving local optima. Thus we cannot use a low evaporation rate, that is aimed to memorize the positions of interesting zones, as we need to track the optimum rather than to find new ones in previously explored zones. In this version, we thus use a high evaporation rate by setting the persistence parameter to $\rho = 0.1$.

4.2.2 Initial visible zone sizes

A particular point with the dynamic simplex used in $DHCIAC_{track}$ is the length of the initial step (i.e. the initial size of the simplex). Indeed, this is a sensitive parameter that must be set according to the "granularity" of the objective function (i.e. the mean distance between two local optima). In $DHCIAC_{track}$, the initial step length of the dynamic simplex is set according to the size of the initial visible zone of an ant. Thus we need to initialize the parameters of the normal distribution $\aleph_{\pi_m,\pi_d}$ used to randomly initialize all ants. If the granularity is unknown, mean and standard deviations are set $\pi_m = \pi_d = \frac{S}{\sqrt[n]{\eta}}$, with S, the search space size (on the smaller dimension) and n, the number of dimensions. This can generally achieve good performance.

5 Experimental results and discussion

We have tested *DHCIAC* for the proposed benchmark set. To avoid any problem due to the choice of an initial population, we performed each test 100 times with a different random seed for each test. To test the efficiency of the algorithm, we measured the mean shortest distance to the global optimum during an optimization procedure (fixed to 100 evaluations of the objective function in our tests). Some of these results are presented in table 2.

The first observation we can make is that *DHCIAC* achieves better performance on slow problems. Furthermore, $DHCIAC_{track}$ is better than

$DHCIAC_{find}$ when we consider a fast problem. For the problems having a "position" target, the *track* version permits a lower mean error, whereas it encounters some difficulties with problems based on "value" target. On the contrary, the *find* version of *DHCIAC* achieves better results for value-based problems than for position-based ones. The problems based on the variations of the number of dimensions and on the variations of both values and positions seem to be difficult to optimize, for both algorithms.

The difficulty of *DHCIAC* for fast problems can be easily understood as it is a population-based algorithm and thus needs a certain number of evaluations to perform a good optimization. In fast problems, this leads to a gap between the movements of the first ants and the last ants that can perturb the algorithm. The $DHCIAC_{track}$ algorithm achieves better performance as it can rapidly go around a moving optimum, due to its dynamic simplex, that needs very few evaluations to proceed toward an optimum.

Moreover, the two strategies that led us to produce two versions of the *DHCIAC* algorithm are shown to be efficient on distinct classes of problems. Indeed, the intensification strategy (implemented with the *track* version) achieves better performance for position-based problems whereas the diversification strategy (the *find* version) is more efficient for value-based problems.

6 Conclusion

We have shown that the *HCIAC* algorithm can be adapted for continuous dynamic optimization problems, leading to the *DHCIAC* algorithm. We have proposed to use two different strategies, based on the idea of adaptive memory programming that points out the notions of intensification and diversification. These strategies have led to two versions of the algorithm: $DHCIAC_{track}$, implementing an intensification behaviour, and $DHCIAC_{find}$, using a diversification one. In addition, we proposed a new benchmark set to test dynamic metaheuristics over a coherent set of test functions covering the main aspects of continuous dynamic optimization.

We have found that the two different strategies are best suited for distinct classes of dynamic problems, namely value-based problems for $DHCIAC_{find}$ and position-based problems for $DHCIAC_{track}$. Moreover, *DHCIAC* seems to be particulary useful for slow dynamic problems with a lot of local optima where the values and the positions of the optima are changing.

As future scope of work, one important issue consists of improving the tuning of the parameters by adapting them to the various classes of problems, before comparing the efficiency of *DHCIAC* to that of the competing metaheuristics.

References

[1] A. Berro. *Optimisation multiobjectif et stratégie d'évolution en environnement dynamique*. PhD thesis, Université Paul Sabatier, Toulouse, De-

cember 2001.

[2] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence, From Natural to Artificial Systems*. Oxford University Press, 1999.

[3] J. Branke. *Evolutionary Optimization in Dynamic Environments*, volume 3 of *Genetic Algorithms and Evolutionary Computation*. Kluwer Academic Publishers, 2001.

[4] A.J. Carlisle. Tracking Changing Extrema with Adaptive Particle Swarm Optimizer. In *ISSCI 2002, World Automation Congress*, Orlando, USA, June 2002.

[5] H.G. Cobb and J.J. Grefenstette. Genetic Algorithms for Tracking Changing Environments. In *Fifth International Conference on Genetic Algorithms*, 1993.

[6] G. Di Caro and M. Dorigo. Ant colonies for adaptive routing in packet-switched communications networks. In A.E. Eiben, T. Bäck, M. Schoenauer, and H.-P. Schwefel, editors, *Proceedings of PPSN-V, Fifth International Conference on Parallel Problem Solving from Nature*, volume 1498 of *Lecture Notes in Computer Science*, pages 673–682, Berlin, Germany, 1998. Springer Verlag.

[7] G. Di Caro and M. Dorigo. AntNet: Distributed stigmergic control for communications networks. *Journal of Artificial Intelligence Research*, 9:317–365, 1998.

[8] J. Dréo and P. Siarry. Un algorithme de colonie de fourmis en variables continues hybridé avec un algorithme de recherche locale. In *5ème Congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision (ROADEF 2003)*, Avignon, France, February 2003.

[9] J. Dréo and P. Siarry. Continuous Interacting Ant Colony Algorithm Based on Dense Heterarchy. *Future Generation Computer Systems*, 20(5):841–856, 2004.

[10] R. Morrison and K. De Jong. A Test Problem Generator for Non-Stationary Environments. In *Congress on Evolutionary Computing*, 1999.

[11] J. A. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*, 7:308–313, 1965.

[12] C.O. Ourique, E.C. Jr Biscaia, and J.C. Pinto. The use of particle swarm optimization for dynamical analysis in chemical processes. *Computers & Chemical Engineering*, 26(12):1783–1793, 15 December 2002.

[13] R. Schoonderwoerd, O. Holland, J. Brutent, and L. Rothkrantz. Ant-based load balancing in telecommunication networks. *Adaptive Behavior*, 5(2):169–207, 1996.

[14] É. D. Taillard, L. M. Gambardella, M. Gendreau, and J.-Y. Potvin. Adaptive Memory Programming: A Unified View of Meta-Heuristics. *European Journal of Operational Research*, 135(1):1–16, 1998.

[15] Q. Xiong and A. Jutan. Continuous optimization using a dynamic simplex method. *Chemical Engineering Science*, 58(16):3817–3828, 2003.

[16] W.S. Yip and T.E. Marlin. Designing plant experiments for real-time optimization systems. *Control Engineering Practice*, 11(8):837–845, August 2003.

[17] W.S. Yip and T.E. Marlin. The effect of model fidelity on real-time optimization performance. *Computers & Chemical Engineering*, 28(1–2):267–280, 15 January 2003.

A Set of test functions

For all the problems listed below, t is considered to be an integer representing the time elapsed.

A.1 OPL: Optimum Position Linear

- *Linear* time function: $T(t) = t$,
- *Easom* structure function:

$$D(x_1, x_2, t) = -\cos(x_1) \cdot \cos(x_2) \cdot e^{-(x_1 - T(t))^2 - (x_2 - T(t))^2}$$

  with $-50 < x_1, x_2 < 50$,
- Target: optimum position,
- Characteristics: continuous, symmetric search space, mono-modal, convex optimum bowl.

A.2 APL: All Position Linear

- *Linear* time function: $T(t) = t$,
- *B1* structure function:

$$\begin{aligned} D(x_1, x_2, t) = \quad & (x_1 - T(t))^2 + 2(x_2 - T(t))^2 \\ & -0.3 \cdot \cos(3\pi - (x_1 - T(t))) \\ & -0.4 \cdot \cos(4\pi - (x_2 - T(t))) \\ & +0.7 \end{aligned}$$

  with $-10 < x_1, x_2 < 10$,
- Target: whole function position,
- Characteristics: continuous, convex, symmetric search space, symmetric function, multi-modal, periodic, several local optima.

A.3 OVP: Optimum Value Periodical

- *Periodical* time function: $T(t) = 8 \cdot \cos(t + 0.5)$,
- *Morrison* structure function:

$$D(x_1, x_2, t) = \sum_{i=1}^{5} \min\left(-H_i + R_i \cdot \left((x_1 - P_{i,1})^2 + (x_2 - P_{i,2})^2\right)\right)$$

  with:

  - $-10 < x_1, x_2 < 10$

- $H = \{1, 3, 5, 7 + T(t), 9\}$
- $R = \{1, 1, 1, 1, 1\}$
- $P = \begin{Bmatrix} 2.5 & -2.5 & -5 & 0 & -4 \\ -2.5 & 2.5 & 5 & 0 & -4 \end{Bmatrix}$

- Target: optimum #4 value,
- Characteristics: continuous, symmetric search space, multi-modal, convex optimum bowl.

A.4 AVP: All Value Periodical

- *Periodical* time function: $T(t) = \cos(0.2 \cdot t)$
- *Shekel* structure function:

$$D(x,t) = T(t) + \sum_{i=1}^{n} \left(\frac{1}{(x - a_i)^T (x - a_i) + c_i} \right)$$

with:

- $x = (x_1, x_2, x_3, x_4)^T$
- $a_i = (a_i^1, a_i^2, a_i^3, a_i^4)^T$ and

i	a_i^T				c_i
1	4	4	4	4	0.1
2	1	1	1	1	0.2
3	8	8	8	8	0.2
4	6	6	6	6	0.4
5	3	7	3	7	0.4
6	2	9	2	9	0.6
7	5	5	3	3	0.3
8	8	1	8	1	0.7
9	6	2	6	2	0.5
10	7	3.6	7	3.6	0.5

- $n = 5$
- $0 < x_i < 10$

- Target: whole function value,
- Characteristics: continuous, multi-modal, several local optima.

A.5 ADL: All Dimension Linear

- *Linear* time function: $T(t) = t$,
- *Rosenbrock* structure function:

$$D(x,t) = \sum_{i=1}^{T(t)} \left(100 \cdot \left(x_i^2 - x_{i+1} \right)^2 + (x_i - 1) \right)$$

with $-5 < x_i < 10$,

- Target: number of dimensions,
- Characteristics: continuous, symmetric search space, quadratic, multi-modal, several local optima.

A.6 O(P+V)P: Optimum Position and Value Periodical

- *Periodical* time function: $T(t) = 8 \cdot \cos(t + 0.5)$,
- *Morrison* structure function:

$$D(x_1, x_2, t) = \sum_{i=1}^{5} \min\left(-H_i + R_i \cdot \left((x_1 - P_{i,1})^2 + (x_2 - P_{i,2})^2\right)\right)$$

  with:

  - $-10 < x_1, x_2 < 10$
  - $H = \{1, 3, 5, (7 + T(t)), 9\}$
  - $R = \{1, 1, 1, 1, 1\}$
  - $P = \begin{Bmatrix} 2.5 & -2.5 & -5 & T(t) & -4 \\ -2.5 & 2.5 & 5 & T(t) & -4 \end{Bmatrix}$

- Target: optimum #4 position and value,
- Characteristics: continuous, symmetric search space, multi-modal, convex optimum bowl.

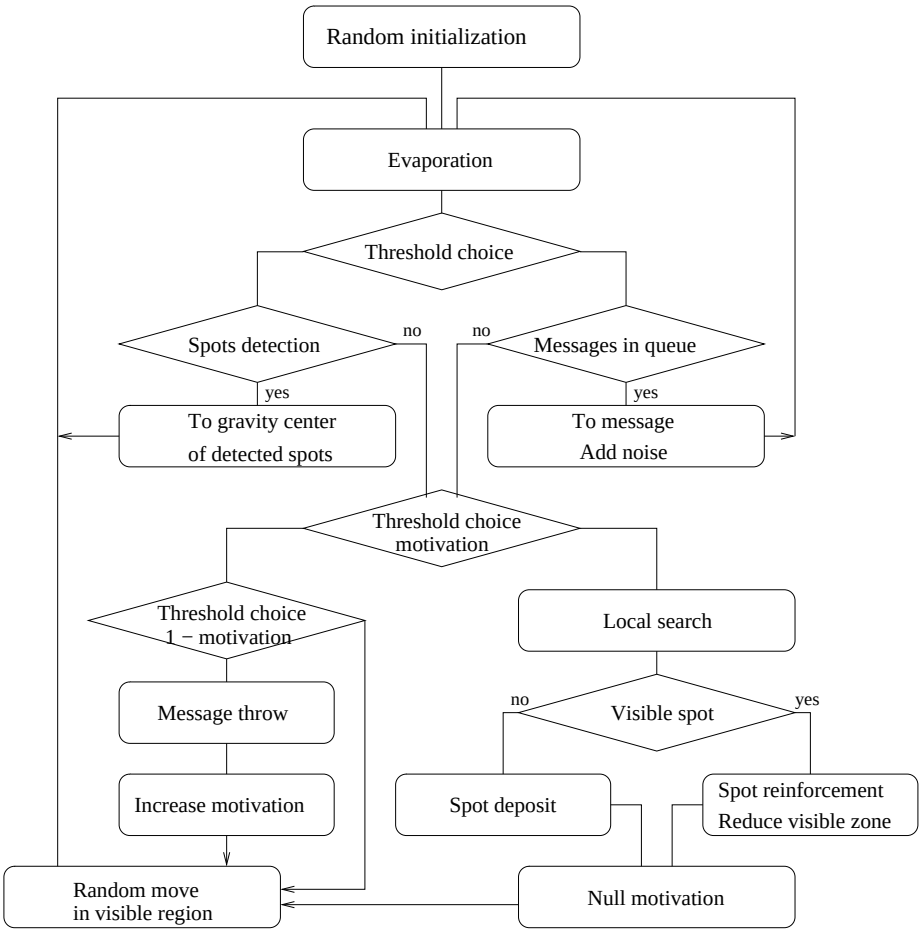

Figure 1: The HCIAC algorithm

Table 1: Sets of ν and η parameters which can achieve a satisfying intensification/diversification behaviour for the *HCIAC* algorithm on the O(P+V)P (see appendix A.6) dynamic problem with two dimensions.

P_i	1	2	3	4	5	6	7	8	9	10
ν	95	93	90	90	50	40	40	34	25	15
η	95	35	10	10	15	8	15	5	3	3

Table 2: Mean and standard deviation of the error for the two versions of *DHCIAC* optimizing different problems. Problem names are abbreviated using the following notations: the first and second letters indicate the target (O: optimum, A: all ; P: position, V: value, D: dimension), the third letter is for the time function (L: linear, P: periodic) and the last letter stands for the speed of the variations (S: slow, F: fast). The values presented are the means for 100 different tests.

	track		find	
Problem	*mean*	std	*mean*	std
OPLF	**0.56**	**0.33**	0.86	0.65
OPLS	***0.48***	***0.32***	*0.79*	*0.59*
APLF	**18.0**	**9.1**	19.9	17.1
APLS	***17.3***	***7.2***	*19.7*	*15.6*
OVPF	8.1	6.2	**5.0**	**4.7**
OVPS	*7.7*	*5.1*	***4.3***	***3.9***
AVPF	17.1	11.1	**9.5**	**8.2**
AVPS	*15.6*	*9.8*	***8.9***	***7.7***
ADLS	10.1	17.3	7.9	12.1
O(P+V)PS	11.2	8.6	10.9	9.1

Algorithm 1 Algorithm of the dynamic simplex.

1. The starting simplex in iteration k is $S_0 = \{x_j\}_{j=1}^{N+1}$ and the objective function values on these vertices are $\{g_j\}_{j=1}^{N+1}$,

2. Sort the vertices of the simplex such that $g1 \geq g2 \geq \ldots \geq g_{N+1}$,

3. Successive reflections. A single point reflection of the first (worst) point of the simplex S_0, i.e., x_1, is operated

$$x_{N+2} = \frac{2}{N} \sum_{j=2}^{N+1} x_j = x_1$$

and one evaluates $g_{N+2} = g(x_{N+2})$. Thus a new simplex $S_1 = \{x_j\}_{j=2}^{N+2}$ is formed. Reflect the first point of this new simplex S_1, i.e., x_2,

$$x_{N+3} = \frac{2}{N} \sum_{j=3}^{N+1} x_j = x_2$$

and obtain the evaluation $g_{N+3} = g(x_{N+3})$. Thus we have the second new simplex $S_2 = \{x_j\}_{j=3}^{N+3}$. Keep reflecting the first point of the newly formed simplex and after the reflection is repeated M times we have $S_M = \{x_j\}_{j=M+1}^{N+M+1}$ and a series of new simplices $\{S_p\}_{p=1}^{M}$.

4. Choose the starting simplex for iteration $k+1$. Calculate the average function value at these simplices $\{S_p\}_{p=1}^{M}$

$$\bar{g}_{S_p} = \frac{1}{N+1} \sum_{j=p+1}^{N+p+1} g_j,\ p = 1, 2, \ldots, M.$$

Select the simplex S_q that satisfies $\bar{g}_{S_q} = \min \left\{ \bar{g}_{S_p} \right\}_{p=1}^{M}$

5. Re-measure the response at point x_{N+1}.

6. If the maximum number of evaluations is reached, stop,
else set S_q as the new starting simplex for the next iteration and return to 2.