Published in *NICST'2013 (New and smart Information Communication Science and Technology to support Sustainable Development), 18-20 September 2013, Clermont Ferrand, France.*

LOOM, an algorithm for finding local optima of expensive functions

Jeremy Riviere*, Rodolphe Le Riche*† and Gauthier Picard*
*Ecole Nationale Supérieure des Mines de Saint-Etienne
St-Etienne, France
†CNRS LIMOS

***Abstract*—Engineering optimization often involves one or many computationally intensive softwares that must be called to calculate the performance of candidate solutions. Despite the calculation cost, it is useful to characterize the global and the local optima. A new algorithm is described here that searches for all the local optima in a reduced number of calls to the true performance function. The algorithm is based on repeated local searches on a metamodel of the true performance function and is called LOOM for LOcal Optima through Metamodels. The local optima are identified as an output of the search. The search distributes computational resources equally among the basins of attraction. Priority is put on local search (intensification) and exploration occurs within a limited budget of calls to the objective function. This article presents the algorithm and describes a first series of tests in two dimensions where a kriging metamodel is used.**

I. Introduction

Numerical optimization is a part of the engineering practice that either aims at designing systems by maximizing performance or at identifying models by minimizing the distance between the model prediction and target data. In both cases, finding local optima is important. In design optimization, as the project advances, new constraints are added which discard some of the possible design options among which the previous global optimum may stand. Some local optima may then become the new global solutions. In model identification, when the model is richly parameterized in comparison to the amount of available data, there is a large, often infinite, number of optima. Describing the local optima improves a lot the understanding of the model.

Many previous works have proposed stochastic heuristics to locate local optima of functions. Some of these works are based on niching strategies in evolutionary algorithms [1], [5], [18]. Others have modified particle swarm optimization algorithms to increase sampling near the local optima [2], [7], [11], [10], [17].

The current algorithm, called LOOM for LOcal Optima through Metamodels, differs from past related work in two ways : firstly, LOOM can be applied to numerically costly optimization problems. Indeed, the most common difficulty in engineering optimization is the numerical cost of evaluating the performance of a candidate solution relatively to the size of the search space. LOOM is sparse in terms of number of calls to the performance functions thanks to the use of a metamodel. Secondly, LOOM provides as an output a list of possible local optima based on the function metamodel. In the literature, either the local optima are not explicitly identified, or they are isolated a posteriori using clustering techniques.

LOOM originates in the work presented in [15], [14], [16]. While basins of attraction were then described by Voronoi cells, LOOM uses the more flexible metamodel to identify and describe them.

Before providing more details, let us introduce some notations. We consider an unconstrained optimization problem,

$$\underset{x\in\mathcal{S}\subset\mathbb{R}^n}{\text{minimize}} \quad f(x) \tag{1}$$

where f is the objective function that quantifies the performance of candidate solutions x in a search space $\mathcal{S}$ (a compact and bounded subset of $\mathbb{R}^n$). In metamodel (or surrogate) based optimization [4], the objective function f is iteratively approximated by a function $\hat{f}^{(t)}$ at time (or iteration) t. The metamodel $\hat{f}^{(t)}$ is built from a set of t optimization variables $\mathbb{X} \in \mathbb{R}^{n\times t}$ and the associated objective function values $\mathbb{F} \in \mathbb{R}^t$. We will refer to $\{\mathbb{X},\mathbb{F}\}$ as either the Design Of Experiments (DOE) or the database. At time t, the initial problem of Eq.(1) can be approximated by

$$\underset{x\in\mathcal{S}\subset\mathbb{R}^n}{\text{minimize}} \quad \hat{f}^{(t)}(x) \tag{2}$$

It should be noted that solving the approximated problem Eq. 2 involves no call to the true objective function f. In the context of expensive optimization, it is considered that solving Eq. 2 has a negligible computational cost.

II. Description of the LOOM algorithm

The LOOM algorithm aims to build a metamodel of the objective function, identify basins of attraction of local optima and share computational resources for search between the basins of attraction. We will call a unit of computational spending for search an *agent*. Each agent will take search actions that are located with respect to a point in the space $\mathcal{S}$, the position of the agent. The agents move across the search space either by local optimization of the metamodel or by stochastic sampling. After each move, an agent calls the true objective function and the metamodel refined. Through local optimizations, agents locate local optima of the metamodel. LOOM keeps only one agent per basin of attraction. If the metamodel is close enough to the true objective function, each agent location is also a function local optimum. This section describes the initialization phase, the agent's behavior, the method to find a new basin and the algorithm's parameters.

A. Initialization and agent's behavior

The first step of the LOOM algorithm consists in creating the Design Of Experiment (DOE): a set of points $\{\mathbb{X}\}$ is

uniformly picked in the search space $\mathcal{S}$, and their objective function values $\{\mathbb{F}\}$ are computed. An initial metamodel $\hat{f}^{(0)}(x)$ is built from these points. In our implementation, we use a kriging metamodel [3]. The number of points of the DOE linearly depends on the number of variables (or dimension, n). In algorithmic notation, this first step is:

Uniformly sample N=$5 \times n$ *points within xmin and xmax*

$$\{\mathbb{X}\} = \begin{bmatrix} x_1^1 & \dots & x_n^1 \\ . & . & . \\ x_1^N & \dots & x_n^N \end{bmatrix};$$

For each $x^i \in \mathbb{X}$*, compute its* f *value*

$$\{\mathbb{F}\} = \begin{bmatrix} f(x^1)) \\ \dots \\ f(x^N) \end{bmatrix};$$

Build the first metamodel
$\hat{f}^{(0)}(x) = buildMetamodel(\mathbb{X}, \mathbb{F});$

A first *agent* is created at the point which has the minimum f value. When created, the agents follow the same cycle at all iterations t:

- make a local optimization at the agent position on the metamodel $\hat{f}^{(t)}$. In our implementation, we use the Nelder-Mead algorithm [9] with a small initial simplex size;
- if the result of the local optimization is *close* to the agent's position, reduce the *close* parameter and **look for a new basin** of attraction on the metamodel;
  - if a new basin is found and it is unoccupied, then compute this new point's f value, update the metamodel and create a new agent in this basin;
  - if no new basin is found, or if a basin is found occupied by another agent, then **explore**, *i.e.* find the point which is the farthest from its neighbours, compute this point's f value and refine the metamodel (stochastic sampling).
- if the result of the local optimization is not close to the agent's position, then
  - if another agent is *almost* at the position of the result of the local optimization, then remove the worst of the current and the other agents
  - else, calculate the true f at the result of the local optimization, if it is less than the value of the current position, move the agent to this new position. Update the metamodel.

In algorithmic notation, the behavior of agent *ag* at position c is:

Local optimization at c at iteration t
$\hat{x}^* = localOptim(\hat{f}^{(t)}(x), c);$
if $\hat{x}^*$ is Close to c **then**
  Reduce the parameter Close
  $Close = Close - \phi;$
  Look for a new basin within the space bounds $xmin$ *and* $xmax$
  $x_b = seekBasin(\hat{f}^t(x), c, xmin, xmax);$
  if $x_b \neq \emptyset \wedge x_b$ is not *Almost* included in the set of agents A **then**
    Create a new agent a at x_b*, add this agent to the set of agents A, compute its f value and update the number of calls*
    $\{\mathbb{X}\} = \{\mathbb{X}\} \cup x_b;$
    $\{\mathbb{F}\} = \{\mathbb{F}\} \cup f(x_b);$
    $A = A \cup a;$
    $Numberofcall = numberofcall + 1;$
    Update the metamodel $\hat{f}(x)$
    $\hat{f}^{(t)}(x) = updateMetamodel(\mathbb{X}, \mathbb{F});$
  else
    Explore (stochastic sampling)
    $x_e = explore(\hat{f}^{(t)}(x), \mathbb{X});$
    x_e *is the farthest point from its neighbours: compute its value with f and update the number of calls*
    $\{\mathbb{X}\} = \{\mathbb{X}\} \cup x_e;$
    $\{\mathbb{F}\} = \{\mathbb{F}\} \cup f(x_e);$
    $Numberofcall = numberofcall + 1;$
    Update the metamodel $\hat{f}^{(t)}(x)$
    $\hat{f}^{(t)}(x) = updateMetamodel(\mathbb{X}, \mathbb{F});$
  end if
else
  Reset the Close parameter
  if $\hat{x}^*$ is *Almost* included in the set of agents A **then**
    Remove the current agent ag
    $A = A \setminus ag;$
  else
    Compute the value of $\hat{x}^*$ *with f and update the number of calls*
    $\{\mathbb{X}\} = \{\mathbb{X}\} \cup \hat{x}^*;$
    $\{\mathbb{F}\} = \{\mathbb{F}\} \cup f(\hat{x}^*);$
    $Numberofcall = numberofcall + 1;$
    Update the metamodel $\hat{f}^{(t)}(x)$
    $\hat{f}^{(t)}(x) = updateMetamodel(\mathbb{X}, \mathbb{F});$
    if $f(\hat{x}^*) < f(c)$ **then**
      Move the agent to this new position
      $c \leftarrow \hat{x}^*;$
    end if
  end if
end if

As the LOOM algorithm is sequential, each agent acts in turn. The stopping condition is a maximum number of calls to the true objective function, f. We set it proportionally to the number of variables.

B. Looking for a new basin

In order to find a new basin, an agent travels across the search space $\mathcal{S}$ following an infinite half straight line originating from its current position. Along this line and according to the parameters *speed* and *acceleration*, the agent picks new points and makes local optimizations (always on the metamodel). A local optimization resulting in a different point than the agent's position means that a new basin of the metamodel has been found.

Create the direction d according to the dimension n
$d = randn(n);$
d normalization

$d = \frac{d}{\sqrt{d^T d}}$;
while The agent is still in bounds **do**
Follow direction d from the point c, according to the speed parameter
$x_{start} = c + speed * d$;
Local optimization at x_{start}
$\hat{x}^* = localOptim(\hat{f}^{(t)}(x), x_{start})$;
if $\hat{x}^*$ is not Close to x_{start} **then**
A new basin has been found
$return \quad \hat{x}^*$;
else
Increase the speed according to the acceleration parameter
$speed = speed * acc$;
end if
end while
No new basin found
$return \quad \emptyset$;

C. Parameters

The eight parameters introduced in the LOOM algorithm are defined according to the number of variables n and to the bounds of the search space (cf. table I).

TABLE I. LOOM ALGORITHM PARAMETERS VALUES

Parameter	Value
Number of points of the DOE (*DOE*)	$5 \times n$
Maximum number of call of f (*callMax*)	$150 \times n$
Proximity parameters	
Between agents (*Close*)	$0.02 \times \|\|xmin - xmax\|\|$
Agents - optima (*Almost*)	$0.01 \times \|\|xmin - xmax\|\|$
Close reduction factor (ϕ)	$\frac{1}{20}(0.02 \times \|\|xmin - xmax\|\|)$
Ray launching parameters	
Speed (*speed*)	$0.02 \times \|\|xmin - xmax\|\|$
Acceleration (*acc*)	1.2

III. EXPERIMENTATION AND RESULTS

A. Test functions

The LOOM algorithm has been tested with a set of standard benchmark optimization problems in two dimensions. Anticipating further tests, we chose functions that can be generalized to any number of dimensions n (excepted Branin).

$$branin(x) = x_2 - \frac{5.1}{4 * \pi^2} * x_1^2 + (\frac{5 * x_1}{\pi} - 6)^2 + 10 * (1 - \frac{1}{8 * \pi}) * \cos(x_1) + 10. \quad (3)$$

$$michalewicz(x) = \sum_{i=1}^{n} \sin(x_j) * (\sin(\frac{j * x_j^2}{\pi}))^{2m} \quad (4)$$

with $m = 2$.

$$rastrigin(x) = 10n + \sum_{i=1}^{n} x_i^2 - 10 * \cos(2\pi * x_i) \quad (5)$$

In the functions above, *Branin* has three global optima. *Michalewicz* has one global optimum and $n! - 1$ local optima. *Rastrigin* has one global optimum and $3^n - 1$ local optima. These functions are plotted in two dimensions in the figure 1. The table II provides the domain of definition of each function and the positions of the global and local optima.

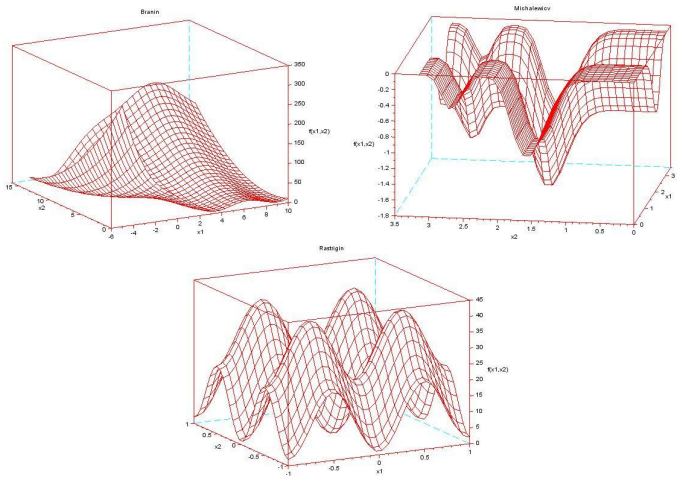

Fig. 1. The first three functions of the benchmark in two dimensions: Branin, Michalewicz and Rastrigin.

We add one function to this benchmark, called the Udder function. Noticing that the functions found in classical test cases for global optimization either were defined for specific dimension numbers or had a rapidly (geometrically) growing number of local optima (e.g., Michalewicz and Rastrigin), we have created the Udder function in an attempt to have a function defined in any number of dimensions but with a given number of local optima. This function is built by "digging holes" (*i.e.* parabola defined by $g_i(x)$) in a larger parabola defined by $\frac{1}{2}x^T x$. The Udder function is thus defined as:

$$udder(x) = \frac{1}{2}x^T x \; - \; \sum_{i=1}^{L} \mathbb{I}_{g_i(x)<0} \; g_i^2(x) \quad (6)$$

where $\mathbb{I}_{g_i(x)<0} = 1$ if $g_i(x) < 0$, $= 0$ otherwise

L is the number of local optima and $g_i(x) = g_i^0 + \frac{1}{2}(x - x_{g_i})^T H_i (x - x_{g_i})$, where x_{g_i} and g_i^0 are the center and the depth of the parabola, respectively.

For the purpose of the experiment, we chose one global optimum at $x_{g_1} = [\frac{1}{2\sqrt{2}}, \frac{1}{2\sqrt{2}}]$, and three local optima defined by $x_{g_2} = [-\frac{1}{2}, 0]$, $x_{g_3} = \beta x_{g_1}$[1] and the optimum of $\frac{1}{2}x^T x$ at $[0, 0]$. The figure 2 shows the Udder function.

B. Experimental setup

A kriging metamodel [13] was chosen to approximate the objective function and the Nelder-Mead algorithm [9] performed the local optimizations. Indeed, the kriging and the Nelder-Mead methods are robust and commonly used in optimization and simulation works [6], [8], [12]. These choices brought us to define the following additional parameters:

[1]The parameter β allows this local optimum to be close to the optimum of $g_1(x)$ without interfering with the $g_1(x)$ parabola.

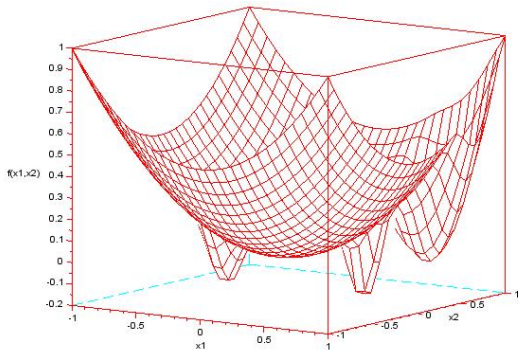

Fig. 2. The Udder function in two dimensions.

TABLE II. DOMAIN DEFINITION AND OPTIMA POSITIONS FOR EACH FUNCTION IN TWO DIMENSIONS.

Function	Domain definition	Optima
Branin	$x_1 \in [-5; 10]$, $x_2 \in [0; 15]$	**- global:** $x_1^* = [-\pi, 12.275]$ $x_2^* = [\pi, 2.275]$ $x_3^* = [9.42478, 2.475]$. **- local:** none.
Michalewicz	$x_i \in [0; \pi]$ $\forall i = 1, 2$	**- global:** $x^* = [2.13755, \frac{\pi}{2}]$. **- local:** $x^* = [2.13755, 2.67830]$.
Rastrigin	$x_i \in [-1; 1]$ $\forall i = 1, 2$	**- global:** $x^* = [0, 0]$. **- local:** $x_1^* = [-1, -1]$ $x_2^* = [-1, 0]$ $x_3^* = [-1, 1]$ $x_4^* = [0, -1]$ $x_5^* = [0, 1]$ $x_6^* = [1, -1]$ $x_7^* = [1, 0]$ $x_8^* = [1, 1]$.
Udder	$x_i \in [-1; 1]$ $\forall i = 1, 2$	**- global:** $x^* = [0.34357, 0.34849]$. **- local:** $x_1^* = [-0.48198, 0]$ $x_2^* = [0.62478, 0.66218]$ $x_3^* = [0, 0]$.

- kriging parameters
  - Regression function: second order polynomial regression function;
  - Correlation function: Gaussian function.
- Nelder-Mead parameters
  - Maximum number of metamodel calls: $50\times$dimension;
  - Simplex size: $0.001 \times ||xmin - xmax||$.

For each function, the LOOM algorithm is run 50 times, starting from different DOEs. The other parameters of the LOOM algorithm are those defined in Section II-C when $n = 2$: in particular, the number of points of the DOE is 10 and the maximum number of call to the true objective function is 300.

C. Results

Our first results are the evolution of the agents number from 10 (the size of the DOE) to 300 calls to the expensive function. The figures 3 and 4 show the evolution of the number of agents with the Branin, Michalewicz, Rastrigin and Udder functions, averaged over the 50 trials.

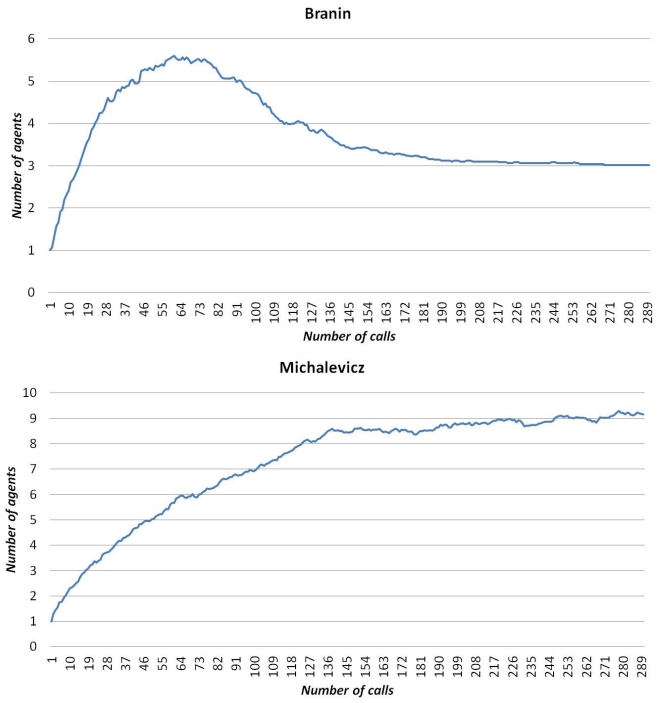

Fig. 3. Average number of agents with the Branin (above) and Michalewicz (below) functions over 50 trials versus number of calls to the true objective function.

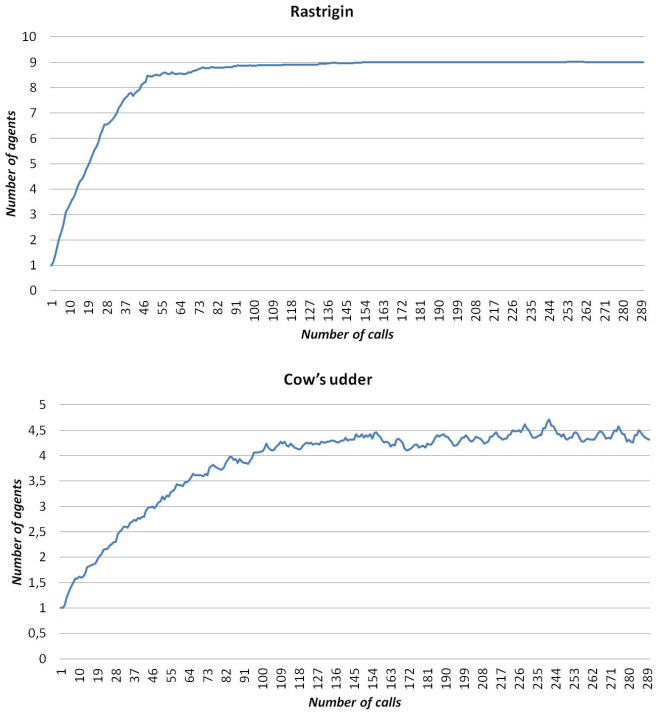

Fig. 4. Average number of agents with the Rastrigin (above) and Udder (below) functions over 50 trials versus number of calls to the true objective function.

These results confirm that the number of agents converges to the number of optima, but the shape of the convergence curve depends on the function. For example, in the case of the

Branin function, the agents number increases fast to 5.5 after 55 calls, then decreases to converge to 3 (the right number of optima) after about 200 calls. Indeed, the metamodel needs a certain number of calls to be close enough to the objective function: during this process, new basins of attraction are discovered but these basins can be temporary artifacts of the metamodel and not exist in the objective function. As the metamodel is refined, some agents previously created are removed when they no longer are at local optima. In the case of the Rastrigin function, the number of agents gradually increases to 9 (the right number of optima) around 150 calls. For this function, the metamodel fits the true function more rapidly, which leads to discovering fewer false basins. The contour plot in figure 5 shows the positions of all the final agents over the 50 trials with the Rastrigin function.

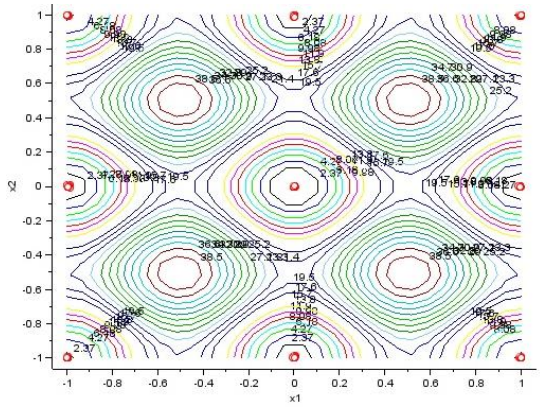

Fig. 5. Positions of all the final agents of the 50 trials with the Rastrigin function.

While the Michalewicz function has 2 local optima in 2D, the average number of agents seems to tend to 9. The specific shape of the Michalewicz function can explain this difference: the Michalewicz function exhibits very flat parts (cf. figure 1) which cannot be accuratly learned by the metamodel in our low number of calls to f. These flat areas will be represented by flat but bumpy surfaces by the kriging model, therefore creating false local optima. The contour plot in figure 6 shows the positions of all the final agents of the 50 trials on the Michalewicz function. We can see that the local optima are correctly occupied by agents (approximately at $x_2 = 2.67$, $x_2 = 1.7$ and $x_1 = 2.13$), but the flat parts of the function are also populated.

To assess the accuracy with which LOOM finds the local and global optima, we have monitored the Euclidean distances between the final positions of the agents and the global and local optima of each function. The figures 7 and 8 show the evolution between 10 and 300 calls of the average accuracy over 50 trials with all our test functions.

These results show that the agents find all the optima and refine their positions as the number of calls increases (and as the metamodel improves). All the curves have the same behaviour: they first decrease rapidly as agents are created and new basins are found, then they decrease slowly to converge to 0. The table III shows, for each function, the percentage of the 50 trials which are successful, *i.e.* accurate according to a distance threshold. This threshold is set to the proximity parameter *Almost*, $0.01 \times ||xmin - xmax||$, equivalent to 1%

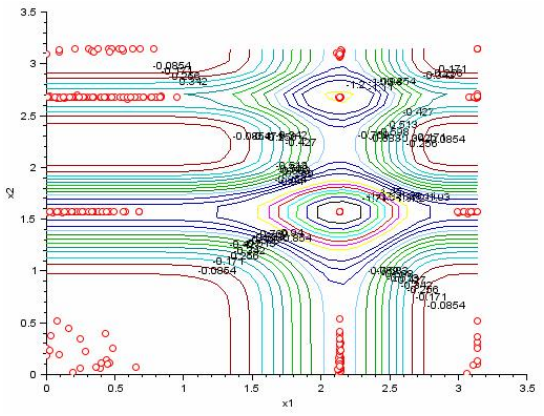

Fig. 6. Positions of the final agents of the 50 trials with the Michalewicz function.

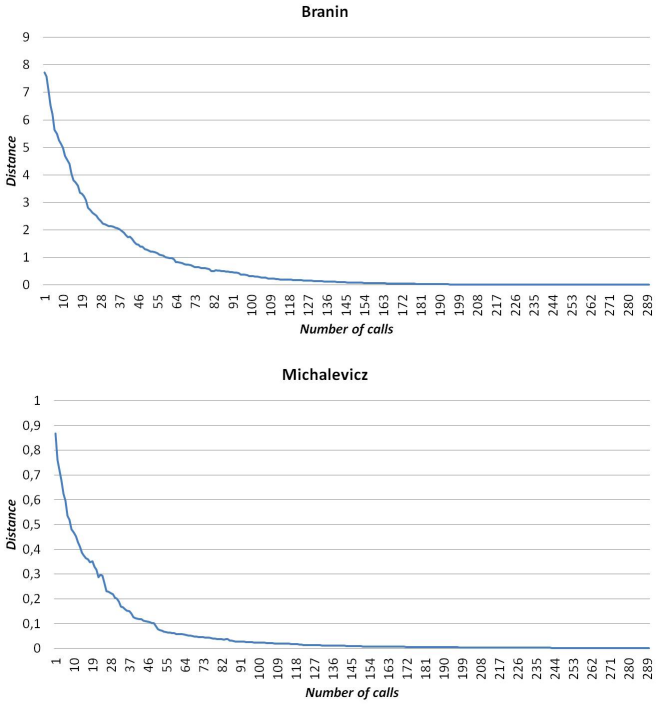

Fig. 7. Average distance between the agents positions and the optima of the Branin (above) and Michalewicz (below) functions, over 50 trials.

of the function's largest diagonal (0.02 for the Rastrigin and Udder functions, 0.15 for the Branin function and 0.0314 for the Michalewicz function).

For all of the functions, 100% of the trials find all the optima with an accuracy better than $0.01 \times ||xmin - xmax||$. The success percentages stay above 75% at higher accuracies for all functions but Rastrigin. To explain this result, we can compare the Rastrigin and the Udder functions, as they have the same bounds. The main difference is that the Rastrigin function has 9 optima, while the Udder function has 4. Having more basins to find divides the number of calls between agents and thus increases the difficulty to improve the accuracy at each optimum.

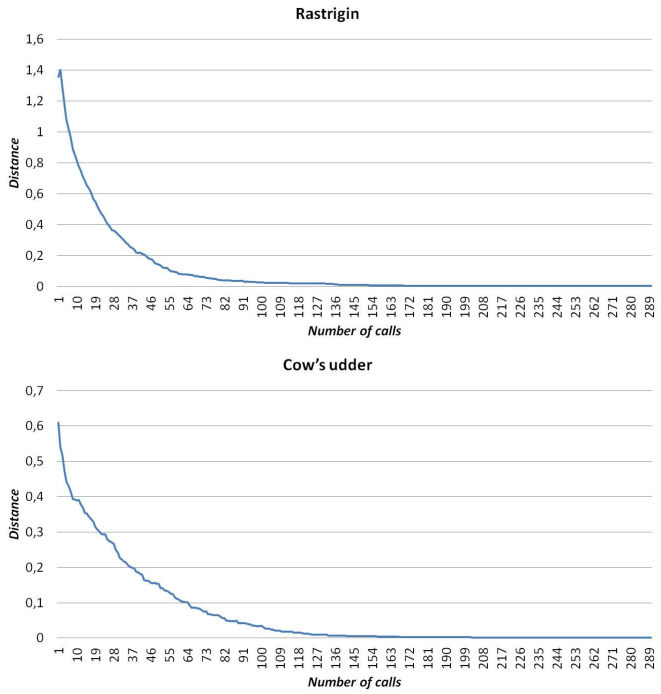

Fig. 8. Average distance between the agents positions and the optima of the Rastrigin (above) and Udder (below) functions, over 50 trials.

TABLE III. FOR EACH FUNCTION, SUCCESS PERCENTAGE OVER 50 TRIALS FOR VARIOUS ACCURACIES.

Function	**distance between agents positions and all optima**		
	$\leq threshold$	$\leq \frac{threshold}{10}$	$\leq \frac{threshold}{20}$
Branin	100% (≤ 0.15)	100% (≤ 0.015)	76% (≤ 0.0075)
Michalewicz	100% (≤ 0.0314)	94% (≤ 0.00314)	84% (≤ 0.00157)
Rastrigin	100% (≤ 0.02)	12% (≤ 0.002)	0% (≤ 0.001)
Udder	100% (≤ 0.02)	100% (≤ 0.002)	78% (≤ 0.001)

IV. CONCLUSIONS

In this paper, we have presented the LOOM algorithm which aims to find the local and global optima of expensive to evaluate functions. This algorithm builds a metamodel of the objective function in order to reduce the number of calls to it, and shares computational resources for search between the basins of attraction (through so-called *agents*). The agents search firstly by local optimization of the metamodel and secondly by stochastic sampling, each search step being concluded by a refinement of the metamodel. The order in which local search and stochastic sampling are performed emphasizes exploitation of past evaluations (or intensification of the search around existing agents) over exploration of the design space.

A first series of experiments was presented with 4 two-dimensional multimodal test functions. The tests show that all local optima are located within an accuracy of 1% of the largest diagonal in the domain for a budget of $150 \times n = 300$ calls to the objective function.

We now intend to test this algorithm in larger number of dimensions and compare it to alternative algorithms for locating local optima.

REFERENCES

[1] David Beasley, David R Bull, and Ralph R Martin. A sequential niche technique for multimodal function optimization. *Evolutionary computation*, 1(2):101–125, 1993.

[2] R Brits, A P Engelbrecht, and F van den Bergh. Locating multiple optima using particle swarm optimization. *Applied Mathematics and Computation*, 189(2):1859–1883, 2007.

[3] Noel A. C. Cressie. *Statistics for Spatial Data*. Wiley, revised edition edition, January 1993.

[4] Jones D.R. A taxonomy of global optimization methods based on response surfaces. *Journal of Global Optimization*, 21:345–383, 2001.

[5] Cem Hocaoglu and Arthur C Sanderson. Multimodal function optimization using minimal representation size clustering and its application to planning multipaths. *Evolutionary Computation*, 5(1):81–104, 1997.

[6] J.\ P.\ C Kleijnen. Kriging metamodeling and simulation: a review. *European Journal of Operational Research*, 192(3):707–716, 2009.

[7] X Li. Adaptively Choosing Neighbourhood Bests Using Species in a Particle Swarm Optimizer for Multimodal Function Optimization. In *Genetic and Evolutionary Computation (GECCO 2004)*, volume 3102 of *Lecture Notes in Computer Science*, pages 105–116, 2004.

[8] M. A. Luersen and R. Le Riche. Globalized Nelder-Mead method for engineering optimization. In *ICECT'03: Proceedings of the third international conference on Engineering computational technology*, pages 165–166, Edinburgh, UK, 2002. Civil-Comp press.

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

[10] Daniel Parrott and Xiaodong Li. Locating and tracking multiple dynamic optima by a particle swarm model using speciation. *IEEE Trans. Evol. Comput*, pages 440–458, 2006.

[11] K E Parsopoulos and M N Vrahatis. Modification of the Particle Swarm Optimizer for locating all the global minima. *Artificial Neural Networks and Genetic Algorithms*, pages 324–327, 2001.

[12] Johan Persson and Johan Ölvander. Comparison of different uses of metamodels for robust design optimization. In *American Institute of Aeronautics and Astronautics*, 2013.

[13] T W Simpson, T M Mauery, J J Korte, and F Mistree. Kriging models for global approximation in simulation-based multidisciplinary design optimization. *AIAA Journal*, 39(12):2233–2241, 2001.

[14] D Villanueva, R Le Riche, G Picard, and R T Haftka. Design space partitioning for optimization using agents. In *14th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, Indianapolis, IN, 2012.

[15] D Villanueva, R Le Riche, G Picard, and R T Haftka. Surrogate-Based Agents for Constrained Optimization. In *14th AIAA Non-Deterministic Approaches Conference*, Honolulu, HI, 2012.

[16] D Villanueva, R Le Riche, G Picard, and R T Haftka. Dynamic design space partitioning for optimization of an integrated thermal protection system. In *9th AIAA Multidisciplinary Design Optimization Specialist Conference*, Boston, MA, 2013.

[17] Shengxiang Yang and Changhe Li. A clustering particle swarm optimizer for locating and tracking multiple optima in dynamic environments. *IEEE Trans. Evol. Comput*, pages 959–974, 2010.

[18] Jun Zhang, De-Shuang Huang, Tat-Ming Lok, and Michael R. Lyu. A novel adaptive sequential niche technique for multimodal function optimization. *Neurocomputing*, 69(16-18):2396–2401, October 2006.