Self-organized Space Partitioning for Multi-Agent Optimization

Diane Villanueva[1,2], Rodolphe Le Riche[2,3], Gauthier Picard[2], and Raphael T. Haftka[1]

[1] *University of Florida, Gainesville, FL, 32611, USA*
[2] *École Nationale Supérieure des Mines de Saint-Étienne, Saint-Étienne, France*
[3] *CNRS UMR 5146*

Abstract. In this paper we explore the use of multi-agent systems to tackle optimization problems in which each point is expensive to get and there are multiple local optima. The proposed strategy dynamically partitions the search space between several agents that use different surrogates to approximate their subregion landscape. Agents coordinate by exchanging points to compute their surrogate and by modifying the boundaries of their subregions. Through a self-organized process of creation and deletion, agents adapt the partition as to exploit potential local optima and explore unknown regions. The overarching goal of this technique is to all local optima rather than just the global one. The rationale behind this is to assign adequate surrogate to each subregion so that (i) optimization is cheaper, (ii) the overall optimization process is not only global in scope but also stabilizes on local optima and (iii) the final partitioning provides a better understanding of the optimization problem.

1 Introduction

Many contemporary applications can be modeled as distributed optimization problems (ambient intelligence, machine-to-machine infrastructures, collective robotics, complex product design, etc.). Optimization processes iteratively choose new points in the search space and evaluate their performances until a solution is found. However, a practical and common difficulty in optimization problems is that the evaluations of new points require expensive computations. For instance, if an agent wants to compute a property of a complex object (e.g. the maximum deflection of an aircraft wing), it may perform a high fidelity computation (e.g. finite element analysis). Therefore, many researchers in the field of optimization have focused on the development of optimization methods adapted to expensive computations. The main ideas underlying such methods are often the use of surrogates, problem decomposition, and parallel computation. The use of surrogates to replace expensive computations and experiments in optimization has been well documented [7, 14, 16, 20]. Moreover, in optimization, a common way to decompose problems is to partition the search space [26, 25]. And, to take advantage of parallel computing, many have proposed strategies for using multiple surrogates in optimization [24, 17, 23, 4]. Besides these techniques for distributing the solving process, multi-agent optimization is an active research field proposing solutions for distinct agents to find cooperatively solution to distributed problems [19]. They mainly rely on the distribution (and decomposition) of the formulation of the problem. Generally, the optimization framework consists in distributing variables and contraints among several agents that cooperate to set values to variables that optimize a given cost function, like in DCOP model [12]. Another approach is to decompose problems or to transform problems in dual problems that can be solved by separate agents [6] (for problems with specific properties like linear problems).

Here, we describe a multi-agent method in which the search space is dynamically partitioned (and not the problem formulation) into subregions in which each agent evolves and performs a surrogate-based continuous optimization. The novelty of this approach comes from the joint use of (*i*) surrogate-based optimization techniques for expensive computation and (*ii*) self-organization techniques for partitioning the search space and finding all the local optima. Coordination between agents, through exchange of points and self-organized evolution of the subregion boundaries allows the agents to stabilize around local optima. Like some specifically designed particle swarm [2, 13, 10] , genetic algorithms [9], or clustering global optimization algorithms (chap. 5 in [21]), our goal is to locate all local optima, but contrarily to these algorithms, our approach sparingly calls the true objective function and constraints. For real-world problems, the ability to locate many optima in a limited computational budget is desirable as the global optimum may be too expensive to find, and because it provides the user with a diverse set of acceptable solutions. Our multi-agent approach further (i) uses the creation of agents for explorating the search space and, (ii) deletes agents to increase efficiency.

In the next section, we provide some background on surrogate-based optimization. In Section 3, we describe the autonomous agents that perform the cooperative optimization process. In Section 4, we present the methods of space partitioning and point allocation that are intended to distribute local optima among the partitions while maximizing the accuracy of the surrogate in a self-organized way –through agent creation and deletion. In Section 5, a constrained optimization example is treated to illustrate our multi-agent approach. Finally, in Section 6 a 6-dimensional problem is tackle using our multi-agent optimizer, before concluding.

2 Surrogate-based optimization

A surrogate is a mathematical function that (i) approximates outputs of a studied model (e.g. the mass or the strength or the range of an aircraft as a function of its dimensions), (ii) is of low computation cost and (iii) aims at predicting new outputs [8]. The set of initial candidate solutions, or points, used to fit the surrogate is called the design of experiments (DOE). Known examples of surrogates are polynomial response surface, splines, neural networks or kriging.

Let us consider the general formulation of a constrained optimization problem,

$$\begin{aligned} &\underset{x \in \mathcal{S} \subset \Re^n}{\text{minimize}} && f(x) \\ &\text{subject to} && g(x) \leq 0 \end{aligned} \tag{1}$$

In surrogate-based optimization, a surrogate is built from a DOE, denoted by $\mathbb{X}$ that consists of sets of the design variables x. For the design of experiments, there are the calculated values of the objective function f and constraints g that are associated with the DOE, which we denote as $\mathbb{F}$ and $\mathbb{G}$, respectively. We will refer to $\mathbb{X}$ and its associated values of $\mathbb{F}$ and $\mathbb{G}$ as a database.

The database is used to construct the surrogate approximation of the objective function $\hat{f}$ and the approximation of the constraint $\hat{g}$. We can approximate the problem in Eq.(1) using the surrogates and formulate the problem as

$$\begin{aligned} \underset{x \in \mathcal{S} \subset \Re^n}{\text{minimize}} \quad & \hat{f}(x) \\ \text{subject to} \quad & \hat{g}(x) \leq 0 \end{aligned} \tag{2}$$

The solution to this approximate problem is denoted $\hat{x}^*$.

Surrogate-based optimization calls for more iterations to find the true optimum, and is therefore dependent on some iteration time t. That is, after the optimum of the problem in Eq.(2) is found, the true values $f(\hat{x}^*)$ and $g(\hat{x}^*)$ are calculated and included in the DOE along with $\hat{x}^*$. At the next iteration, the surrogate is updated, and the optimization is performed again. Therefore, we denote the DOE at a time t as $\mathbb{X}^t$ and the associated set of objective function values and constraint values as $\mathbb{F}^t$ and $\mathbb{G}^t$, respectively. The surrogate-based optimization procedure is summarized in Algorithm 1 (which also refers to a global optimization procedure in Algorithm 2).

Algorithm 1: Overall surrogate-based optimization

```
t = 1 (initial state)
while $t \leq t^{max}$ do
    Build surrogates $\hat{f}$ and $\hat{g}$ from $(\mathbb{X}^t, \mathbb{F}^t, \mathbb{G}^t)$
    Optimization to find $\hat{x}^*$ (see Algorithm 2)
    Calculate $f(\hat{x}^*)$ and $g(\hat{x}^*)$
    Update database $(\mathbb{X}^{t+1}, \mathbb{F}^{t+1}, \mathbb{G}^{t+1}) \cup (\hat{x}^*, f(\hat{x}^*), g(\hat{x}^*))$
    $t = t + 1$
```

Algorithm 2: Constrained optimization procedure

```
input  : $\hat{f}, \hat{g}, \mathbb{X}^t, \mathcal{L}$
output: $\hat{x}^*$
$\hat{x}^* \leftarrow \underset{x \in \mathcal{L}}{\text{argmin}} \hat{f}(x)$ subject to $\hat{g}(x) \leq 0$
if $\hat{x}^*$ is near $\mathbb{X}^t$ or out of the search domain $\mathcal{L}$ then
    $\hat{x}^* \leftarrow \underset{x \in \mathcal{S}}{\text{argmax}}$ distance$(\mathbb{X}^t)$
```

3 Agent Optimization Behavior

As stated in the introduction, our approach consists in splitting the space in subregions and assigning agents to each of these subregions as presented in Fig. 1. Therefore, Algorithm 1 can be thought of as the procedure followed by a single agent to find one point, that will be repeating until termination. But, each agent is restricted to only a subregion of the design space, i.e., $\mathcal{S}$ is replaced by a part of $\mathcal{S}$. The rationale behind

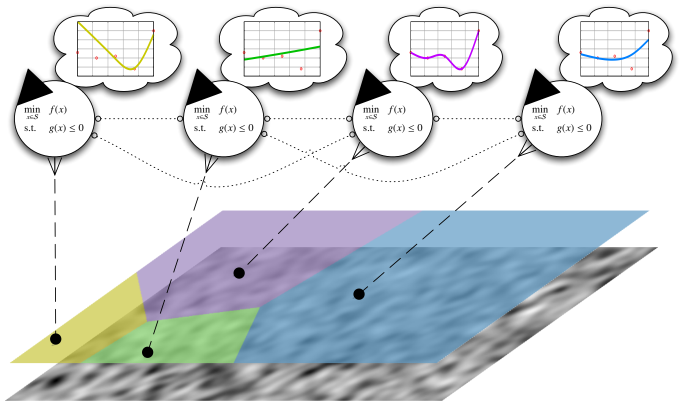


Fig. 1. Multi-agent System overview: agents perform surrogate-based optimization in different subregions of the partitioned search space based on personal surrogates (dashed l.) and exchange points with their direct neighbors (dotted l.)

this idea is that each agent has an easier optimization subproblem to solve because it searches a smaller space, which we denote as $\mathcal{P}_i$ for the ith agent. Each agent must consider only the points in its subregion, which are available in its internal database $(\mathbb{X}^t, \mathbb{F}^t, \mathbb{G}^t)_i$. The subregion of an agent is defined by the position of its center c. A point in the space belongs to the subregion with the nearest center, where the distance is the Euclidean distance. This creates subregions that are Voronoi cells [1]. The choice of where to place the center is discussed in the next section.

Figure 1 illustrates the partition of a two-dimensional space into four subregions for four agents, which requires four centers. In this example, we place the centers randomly. The self-organized process that build space partitions will be discussed in Section 4.

The procedure of a single agent is given in Algorithm 3. Assuming that subregions are defined, each agent fits several surrogates it knows (as many different ways to approximate) and chooses the one that maximizes the accuracy in its subregion (line 5-9). To avoid ill-conditioning, if more points are needed than are available to an agent, the agent asks neighboring agents for points. The neighboring agents then communicate the information associated with these points (lines 6–8). We define the best surrogate as the one with the minimum cross-validation error, the partial prediction error sum of squares $PRESS_{RMS}$. This is found by leaving out a point, refitting the surrogate, and measuring the error at that point. The operation is repeated for p points in the agent's subregion (disregarding any points received from other agents) to form a vector of the cross-validation errors e_{XV}. The value of $PRESS_{RMS}$ is then calculated by

$$PRESS_{RMS} = \sqrt{\frac{1}{p} e_{XV}^T e_{XV}} \tag{3}$$

Once the agents have chosen surrogates (line 9), the optimization is performed by an internal optimizer to solve the problem in Eq.(2) inside the subregion (line 10). If the optimizer gives an infeasible point (i.e., the point does not satisfy the constraint in

Eq.(2) or is out of the subregion) or repeats an existing point, the agent then explores in the subregion. To explore, the agent adds a point to the database that maximizes the minimum distance from the points already in its internal database (see Algorithm 2). The true values f and g of the iterate are then calculated (line 11), and $(\hat{x}^*, f(\hat{x}^*), g(\hat{x}^*))$ is added to the internal database (lines 11–12).

Algorithm 3: Agent i optimization in its subregion.

```
1  t = 1 (initial state)
2  while $t \leq t^{max}$ do
3      Update $\mathcal{P}_i = \{x \in \mathcal{S}$ s.t.$\|x - c_i\|^2 \leq \|x - c_j\|^2 ,\ j \neq i\}$
4      Update internal database from the new space partition
5      Build surrogates $\hat{f}$ and $\hat{g}$ from $(\mathbb{X}^t, \mathbb{F}^t, \mathbb{G}^t)_i$
6      if Not sufficient number of points in internal database to build a surrogate then
7          Get points from other agents closest to $c_i$
8          Build surrogates
9      Choose best surrogate based on partial $PRESS_{RMS}$ error
10     Optimization to find $\hat{x}^*$ [with Algorithm 2$(\hat{f}, \hat{g}, \mathbb{X}^t, \mathcal{P}_i)$]
11     Calculate $f(\hat{x}^*)$ and $g(\hat{x}^*)$
12     $(\mathbb{X}^{t+1}, \mathbb{F}^{t+1}, \mathbb{G}^{t+1})_i \leftarrow (\mathbb{X}^t, \mathbb{F}^t, \mathbb{G}^t)_i \cup (\hat{x}^*, f(\hat{x}^*), g(\hat{x}^*))$
13     Update center $c_i$ (see Section 4.1)
14     Check for merge, split or create (see Section 4.2)
15     $t = t + 1$
```

4 Self-Organized Partitioning

Previous section expounds the cooperative optimization process performed by agents in a pre-partitioned space. However, the partitioning strongly depends on the topology of the space. Therefore, as a part of the cooperative optimization process, we propose a self-organizing mechanism to partition the space which adapts to the search space. By self-organizing, we mean that agents (and therefore subregions) will be created and deleted depending on the cooperative optimization process. Agents will split when points are clustered inside a single region (*creation*), and will be merged when local optima converge (*deletion*).

4.1 Moving Subregions' Centers

The method of space partitioning we propose focuses on moving the subregions' centers to different local optima. As a result, each agent can choose a surrogate that is accurate around the local optimum, and the agent can also explore the subregion around the local optimum. At the beginning of the process, only one agent exists and is assigned to the whole search space. Then it begins optimization by choosing a surrogate, fitting it and optimizing on this surrogate. As a result the agent computes a new point $\hat{x}^{*t-1}$. Then, the center of the subregion is moved to the "best" point in the subregion in terms of feasibility and objective function value (line 13). This is done by comparing the center at the last iteration c^{t-1} to the last point added by the agent $\hat{x}^{*t-1}$. The center is moved to the last point added by the agent if it is better than the current center. Otherwise, the center remains at the previous center. For convenience, in comparing two points x_m and x_n, we use the notation $x_m \succ x_n$ to represent x_m "is better than" x_n. For two centers, instead of points x we would consider the centers c. The conditions to determine the better of two points are given in Algorithm 4.

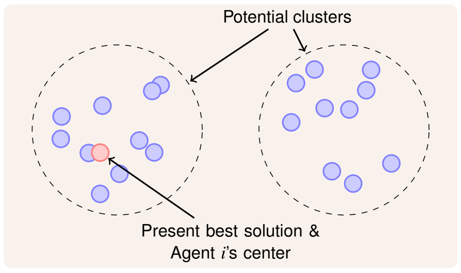
Algorithm 4: Algorithm to determinine if, for two points x_m and x_n, x_m "is better than" x_n ($x_m \succ x_n$) and vice versa.

```
input : $f(x_m), f(x_n), max(g(x_m)), max(g(x_n))$
if $max(g(x_m)) \leq 0$ & $max(g(x_n)) \leq 0$ then
    // both points are feasible
    if $f(x_m) \leq f(x_n)$ then $x_m \succ x_n$
    else $x_n \succ x_m$
else if $max(g(x_m)) \leq 0$ & $max(g(x_n)) > 0$ then
    // only $x_m$ is feasible
    $x_m \succ x_n$
else if $max(g(x_m)) > 0$ & $max(g(x_n)) \leq 0$ then
    // only $x_n$ is feasible
    $x_n \succ x_m$
else if $max(g(x_m)) > 0$ & $max(g(x_n)) > 0$ then
    // none is feasible
    if $max(g(x_m)) \leq max(g(x_n))$ then
        $x_m \succ x_n$
    else
        $x_n \succ x_m$
```

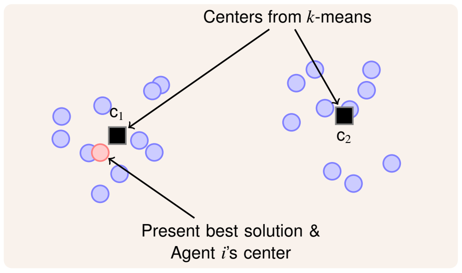
4.2 Merge, Split and Create Subregions

Once an agent has added a new point in its database (line 12) and moved its center to the best point (line 13), it will check whether to split, or to merge with other ones (line 14). Merging agents (and their subregions) prevents agents from crowding the same area, allowing one agent to capture the behavior in a region. Splitting an agent is a way to explore the space as it refines the partitioning of the space in addition to the search that each agent can perform in its subregion. Split and merge occurs at the end of each iteration (line 14): agents are first merged (if necessary), the points belonging to the merged agent(s) are distributed to the remaining agents based on distance from the center of the remaining agents' subregions, and then each remaining agents examines to determine whether to split or not.

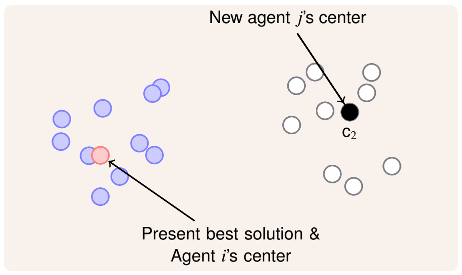
Merge Converging Agents Agents are merged (deleted) if the centers of the agents' subregions are too close as measured by the Euclidean distance between the centers. We measure the minimum Euclidean distance between two centers as a percentage of the maximum possible Euclidean distance between points in the design space. When



(a) Potential clusters within a subregion



(b) Clusters from k-means



(c) Final clusters after moving centers to present best solution and nearest data point

Fig. 2. Illustration of process used to create an agent j given points in a single agent i's subregion.

examining the agents, the agent with the center with the lowest performance is deleted. For example, for agents 1 and 2, if $c_1 > c_2$, agent 2 is deleted. Before deletion, the deleted agent distributes its internal database points to closest neighbors.

Split Clustered Subregions It is desirable to create an agent if it is found that points are clustered in two separate areas of a single agent's subregion, as illustrated in Fig.2(a). Such a situation can occur if there are two optima in a subregion.

Agents are created by using k-means clustering [5] for two clusters ($k = 2$) given the points in the subregion, where the initial guesses of the centers are the present best solution (the current center) and the mean of the dataset. Since k-means clustering gives centers that are not current data points as illustrated in Fig.2(b), we move the centers to available data points to avoid more calls to evaluate the expensive functions. This is done by first measuring the distance of the centers from k-means to the present best solution, and moving the closest center to the present best solution, as we want to preserve this solution. For the other center, we measure the distance of the current data points to the other center, and make the closest data point the other center. The final clustering is illustrated in Fig.2(c). The result is a new agent with a center at an already existing data point, where the creating agent retains its center at its present best solution.

This final clustering is validated using the mean silhouette value of the points in the subregion. The silhouette, introduced by Rousseeuw [15], is used to validate the number of clusters, by providing a measure of the within-cluster tightness and separation from other clusters for each data point i for a set of points. The silhouette value for each point is given as

$$s(i) = \frac{b(i) - a(i)}{max\{a(i), b(i)\}} \tag{4}$$

where a_i is the average distance between point i and all other points in the cluster to which point i belongs, and b_i is the minimum of the average distances between point i and the points in the other clusters. The values of s_i range from -1 to 1. For s_i near zero, the point could be assigned to another cluster. If s_i is near -1, the point is misclassified, and, if all values are close to 1, the data set is well-clustered. The average silhouette of the data points is often used to characterize a clustering. In this paper, we accept the clustering if all s_i are greater than 0 and the average value of the silhouette is greater than some value.

Create New Agents The agents may reach a point where there is no improvement made in several iterations. For example, this can occur when each agent has located the best point in its subregion, the area around each best point is populated by points, each agent is driven to explore for several iterations, and no other potential local optima are located. This can also occur at early iterations in which the surrogates are not well-trained in the subregion. In order to improve exploration, a new agent is created in the design space when there is no improvement for n iterations (i.e., the centers of the subregions have not moved for n iterations). We call this parameter the *stagnation threshold*. To create a new agent, a new center is created at an already existing data point that maximizes the minimum distance from the already existing centers, thus forming a new agent. The design space is then repartitioned.

5 Illustrative Example

In this section, an example is provided to illustrate the effectiveness of the method described in the previous sections. The example is a two dimensional problem with a single constraint taken from Sasena et al. [18]. The feasible regions are disconnected and cover approximately 3% of the design space. The objective is quadratic and the single constraint is the Branin test function [3], and is referred to as *newBranin*. This problem is shown in Eq.(5).

$$\begin{aligned}
\underset{x}{\text{minimize}} \quad & f(x) = -(x_1 - 10)^2 - (x_2 - 15)^2 \\
\text{subject to} \quad & g(x) = \left(x_2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6\right) + \cdots \\
& 10\left(1 - \frac{1}{8\pi}\right)cos(x_1) + 10 - 2 \leq 0 \\
& -5 \leq x_1 \leq 10 \\
& 0 \leq x_2 \leq 15
\end{aligned} \tag{5}$$

The contour plot in Fig. 3 shows three disconnected feasible regions. These feasible regions cover approximately 3% of the design space. The global optimum is located at $x_1 = 3.2143$ and $x_2 = 0.9633$. Local optimum A is located at (9.2153,1.1240) and local optimum B is located at (-3.6685,13.0299).

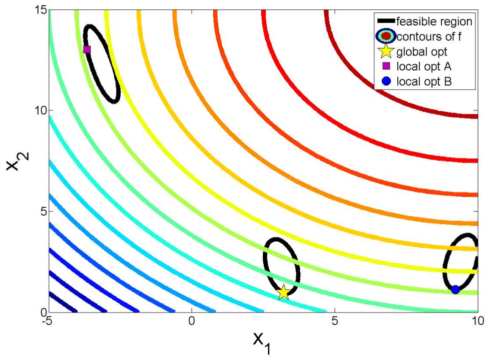


Fig. 3. Contour plot of f for the *newBranin* problem.

5.1 Experimental Setup

In this example, both the objective function f and constraint g were considered to be expensive and were thus approximated with surrogates. The six possible surrogates, which include response surfaces and kriging, are described in Table 1. From this set, each agent chose the best surrogate based on $PRESS_{RMS}$. The set of surrogates and the minimum number of points used to fit each surrogate are provided in Table 1.

Table 1. Surrogates considered in this study

ID	Description	# of pts for fit
1	Linear response surface	
2	Quadratic response surface	1.5 * # of coefficients
3	Cubic response surface	
4	Kriging (quadratic trend)	
5	Kriging (linear trend)	1.5 * # coefficients for QRS
6	Kriging (constant trend)	

For all results, the parameters in Table 2 were used to solve the problem. These parameters include maximum number of agents (e.g. the maximum number of computers available in a cluster), parameters that dictate how close points and centers can be, and parameters that define if a new agent should be created. Considering the fact that function evaluations are expensive, we also fixed a **computation budget** to 132 function evaluations. Beyond this number, the system stops: this is our only termination criterion. Finally, we start the multi-agent system with only one single agent able to split and merge with time.

Table 2. Multi-Agent Parameters

Parameter	Value
Max # of function evaluations	132
Max # of agents	6
Initial/Min # of agents	1
Min distance between agent centers	10% of max possible distance in space
Minimum distance between points	1e-3 (absolute for each dimension)
Min average silhouette	0.4
Min # of points in each agent after creation	4
Stagnation threshold	3

The success and efficiency of the multi-agent approach is compared to a single agent system which performs a classical surrogate-based optimization procedure as described in Algorithm 1. However, this *single agent* is unable to perform dynamic partitioning and optimizes over the whole space. This single agent has also a computation budget of 132 calls to the expensive function. The single agent configuration represents a good standard (containing a state-of-the-art surrogate-based optimizer) to compare our multi-agent optimizer.

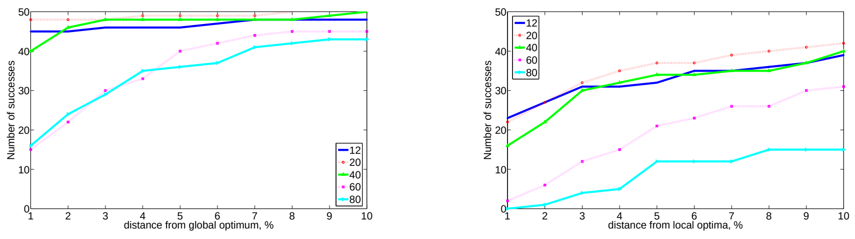
In each case, (multi- or single agent), as to evaluate the capability of the algorithms to explore the search space, we also ran several experiments for different DOE sizes (from 12 to 80) that still account for the number of function evaluations. Therefore, for a larger initial DOE, the system runs less steps. For each of the cases that were studied, the results shown are the median of 50 repetitions (i.e, 50 different initial DOEs). The local optimization problems were solved with a sequential quadratic programming (SQP) algorithm [11]. DOE are obtained using Latin Hypercube sampling.

5.2 Successes to locate optima

The number of successes in locating a feasible solution that is within some percentage distance from the optima are displayed in Fig. 4 for the single agent case and Fig. 5 for the multi-agent case. This distance is the Euclidean distance between a true optimum and the closest solution divided by the maximum possible Euclidean distance between two points in the design space.

It was observed that the single agent was fairly successful overall (except for DOE 60 and 80) at finding a solution near the global optimum (48 successes at a 1% distance from the optimum with DOE 20). However, it was less successful at locating solutions near to all of the optima, with only 40 successes at distance of 10% from the optima with DOE 40.

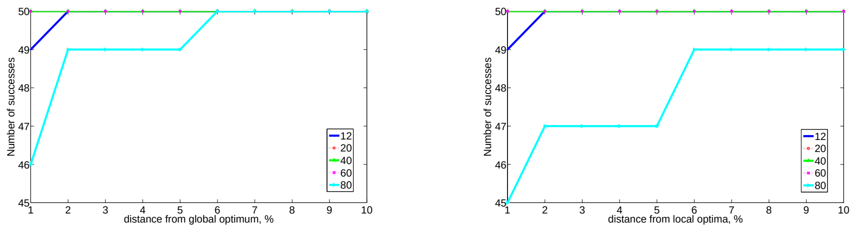
On the contrary, the multi-agent system performs better at locating all optima. The global optimum is found more than 50 times at 1% distance (except for DOE 12 and



(a) Number of successes to locate global optimum

(b) Number of successes to locate all local optima

Fig. 4. For a single agent, (a) number of successes in locating the global optimum versus distance from the optimum, (b) number of successes in locating all the local optima versus distance from the optima



(a) Number of successes to locate global optimum

(b) Number of successes to locate all local optima

Fig. 5. For a multi-agent system, (a) number of successes in locating the global optimum versus distance from the optimum, (b) number of successes in locating all the local optima versus distance from the optima

80). Successes at finding all local optima are also far higher than the single agent case: 50 successes at 1% distance for DOEs except 12 (50 times at 2%) and 80.

It was observed that the global optimum was found by the agent with the smallest DOE (12), followed by local optimum A. Local optimum B was the most difficult to find by the single agent. In the multi-agent case, the figures show that the case with starting with the smallest DOE of 12 points was also overall the most efficient at locating each optimum.

5.3 Agent Dynamics and Efficiency

The particularity of our multi-system is to create and delete agents at runtime. Therefore, Fig. 6 shows the dynamics of the number of agents as a function of the time. Let us note, that each curve ends at a different iteration depending on the DOE size and the number of agents, since each agent is calling the expensive function at each iteration. As we can see, the number of agent stabilized around 3 and 5 agents. Successful cases have more than 3 agents which explains the capability for the multi-agent system to

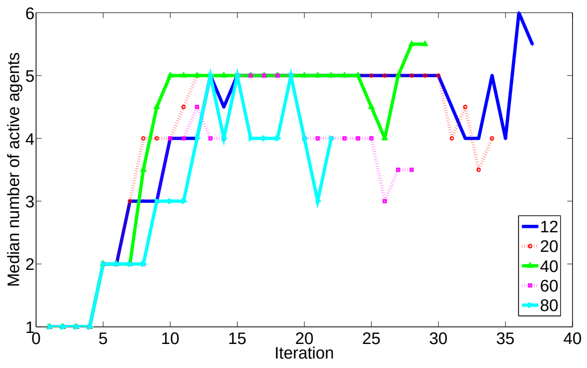


Fig. 6. For a multi-agent system, the median number of agents as a function of time

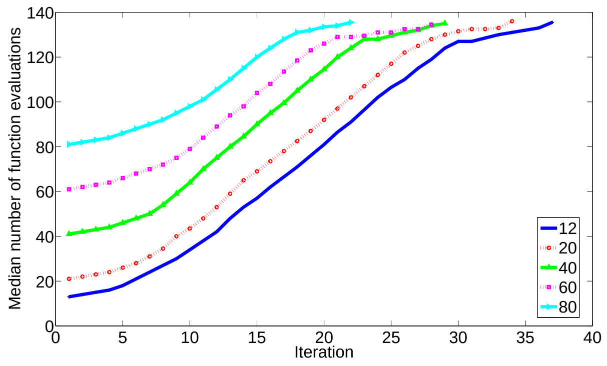


Fig. 7. For a multi-agent system, the median number of function evaluations

locate all the optima. Moreover, the median number of agents is also a good indicator of the potential of parallelization we can expect to gain time in the optimization task. In this example, we can expect diminishing the time up to 75%.

By comparing the single agent to the multi-agent system by function evaluations it seems that the single agent system is as efficient as the multi-agent system. However, looking solely at function evaluations does not reflect how we can parallelize the agents. Therefore, we can look at the number of function evaluations as a function of the number of iterations, which is correlated to wall clock time. For example, the single agent seems to have located all optima around 60 function evaluations with an initial DOE of 12, whereas the multi-agent system does around 70. For the multi-agent system, this 70 function evaluations is around 20 iterations (see Fig. 7), which we roughly equate to 20 parallel function evaluations. In this way, the multi-agent system is 3 times more efficient than the single agent.

5.4 Effect of the size of the initial DOE

On all the presented results, a general trend characterizes the effect of the size of the initial DOE: the multi-agent system is more efficient with small DOEs. This underlines that our multi-agent behavior is better in exploring the search space than the sampling method we used.

To sum up, the multi-agent system is far more efficient than the standard case (single agent) in *(i)* locating the global optimum and in *(ii)* locating both local optima (at small distances and low g values). These show our agent behavior and self-organized partitioning techniques ensure that:

1. optimization is cheaper, since agent can execute concurrently, step by step,
2. the overall optimization process is not only global in scope but also stabilizes on local optima,
3. the final partitioning provides a better understanding of the optimization problem, since each agent is using a different surrogate to optimize.

6 Six-Dimensional Example

In this section, we examine the six-dimsenional Hartman function (Hartman6) that is often used to test global optimization algorithms.

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f_{hart}(x) = -\sum_{i=1}^{q} a_i exp\left(-\sum_{j=1}^{m} b_{ij}(x_j - d_{ij})^2\right) \\ \text{subject to} \quad & 0 \leq x_j \leq 1, j = 1, 2, \ldots, m = 6 \end{aligned} \tag{6}$$

In this instance of Hartman6, $q = 4$ and $a = \begin{bmatrix} 1.0 & 1.2 & 3.0 & 3.2 \end{bmatrix}$ where

$$B = \begin{bmatrix} 10.0 & 3.0 & 17.0 & 3.5 & 1.7 & 8.0 \\ 0.05 & 10.0 & 17.0 & 0.1 & 8.0 & 14.0 \\ 3.0 & 3.5 & 1.7 & 10.0 & 17.0 & 8.0 \\ 17.0 & 8.0 & 0.05 & 10.0 & 1.0 & 14.0 \end{bmatrix}$$

$$D = \begin{bmatrix} 0.1312 & 0.1696 & 0.5569 & 0.0124 & 0.8283 & 0.5886 \\ 0.2329 & 0.4135 & 0.8307 & 0.3736 & 0.1004 & 0.9991 \\ 0.2348 & 0.1451 & 0.3522 & 0.2883 & 0.3047 & 0.3047 \\ 0.4047 & 0.8828 & 0.8732 & 0.5743 & 0.1091 & 0.0381 \end{bmatrix}$$

As we wish to locate multiple optima, we modified Hartman6 to contain 4 distinct local optima by "drilling" 2 additional Gaussian holes at two locations to form two local optima, in addition to the global optimum and 1 local optimum provided in the literature [22]. The modified Hartman6 function is

$$f(x) = f_{hart}(x) - 0.52\phi_1(x) - 0.18\phi_2(x) \tag{7}$$

where the mean and standard deviation associated with ϕ_1 and $\mu_1 = \begin{bmatrix} 0.66 & 0.07 & 0.27 & 0.95 & 0.48 & 0.13 \end{bmatrix}$ and $\sigma = 0.3$ (all directions), respectively. The mean and standard deviation associated with ϕ_2 and $\mu_2 = \left[0.87\ 0.52\ 0.91\ 0.04\ 0.95\ 0.55\right]$ and $\sigma_2 = 0.25$ (all directions), respectively. The optima are displayed in Table 3. To obtain an approximate measure of the size of the basins of attraction that contain the optima, we measured the percentage of local optimization runs that converged to each optimum. To do this, twenty-thousand points were smapled using Latin Hypercube sampling and the local optimization was performed using an SQP algorithm. The percentage of starts that converged to an optima is also a measure of difficulty to locate the optimum as it gives an indication of how narrow a basin is in comparison to other basins. Therefore, it was expected that Local 2 would be the most difficult optimum to locate by the agents.

Table 3. Modified Hartman6 optima and the percentage of runs that found each optimum with multiple starts and a SQP optimizer

Optimum	f	x	percentage of runs
Global	-3.33	$\left[0.20\ 0.15\ 0.48\ 0.28\ 0.31\ 0.66\right]$	50.4
Local 1	-3.21	$\left[0.40\ 0.88\ 0.79\ 0.57\ 0.16\ 0.04\right]$	21.1
Local 2	-3.00	$\left[0.87\ 0.52\ 0.91\ 0.04\ 0.95\ 0.55\right]$	8.7
Local 3	-2.90	$\left[0.64\ 0.07\ 0.27\ 0.95\ 0.48\ 0.13\right]$	19.8

6.1 Experimental Setup

The modified Hartman6 function follows the same experimental setup as the *newBranin* problem. However, instead of using polynomial response surfaces and kriging surrogates, we restricted ourselves to the kriging surrogates only (surrogate IDs 4 to 6 in Table 1). Since there are no nonlinear constraints, only the objective function is approximated by surrogates. We fix the computational budget at 400 total function evaluations, including the evaluations needed to form the initial DOE, and this is used as the termination criterion. The initial DOE size was varied at 35, 56, and 100 points. For all results, the parameters in Table 4 were used to solve the problem.

Table 4. Multi-Agent Parameters for the modified Hartman6 function

Parameter	Value
Max # of function evaluations	400
Max # of agents	8
Initial/Min # of agents	1
Min distance between agent centers	10% of max possible distance in space
Minimum distance between points	1e-3 (absolute for each dimension)
Min average silhouette	0.25
Min # of points in each agent after creation	4
Stagnation threshold	3

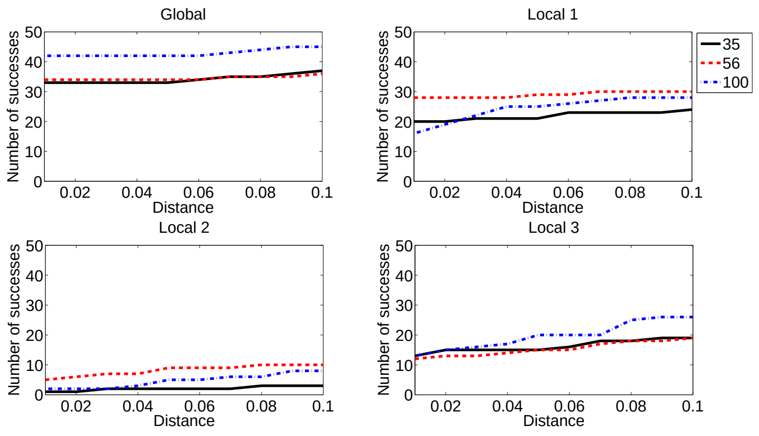


Fig. 8. For a single agent, number of successes in locating each optimum.

6.2 Successes to locate optima

For 50 repetitions, the success in locating each optimum with a single agent and a multi-agent system is shown in Figs. 8 and 9. It was observed that the single agent had fewer successes compared to the multi-agent case. In both the single and multi-agent cases, Local 2 was the optimum that was the most difficult to locate with less than 10 successes with a single agent and 32 successes with a multi-agent system.

6.3 Agent Efficiency and Dynamics

The median objective function value of the solution closest to each optimum is shown in Fig.10. For the global optimum and Local 1, it was observed that the efficiency is nearly equal in the single and multi-agent cases. It was also observed that the smaller DOEs required fewer function evaluations to find these optima. For Local 2 and Local 3, the multi-agent system has a clear advantage in finding solutions with objective function near the true optimum value. While it is clear for Local 3 that smaller DOEs are more efficient in locating the optimum, there is no clear relationship between DOE size and efficiency. Recall that Local 2 was expected to be the most difficult optimum to find judging by the small percentage of times it was found with multiple starts with the SQP optimizer. These results confirm that exploration is required to locate Local 2, and the multi-agent system, in which exploration is an inherent feature, is more capable of locating this optimum.

Figure 11 shows the median number of agents. While up to 8 agents could be created, it was observed that the median number of agents stabilized around 4. This is because as all 4 optima are located, new agents are created and but are then deleted as they converge to the basins of attraction of the optima.

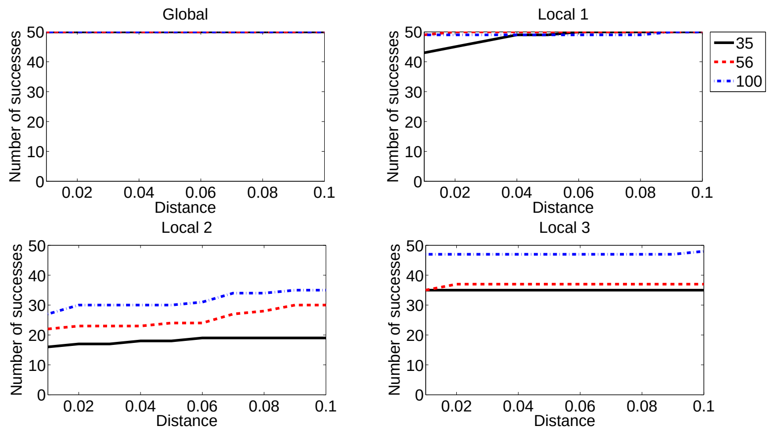


Fig. 9. For a multi-agent, number of successes in locating each optimum.

7 Conclusions and Future Work

This paper introduced a multi-agent technique for optimization that dynamically partitions the optimization variable search space as to find all the local optima. The centers of the agents' subregions moved to stabilize around optima, and agents were created and deleted at run-time as a means of exploration and efficiency, respectively.

Applying our method to two examples (2D and 6D), it was empirically showed that a single agent (i.e., a classical surrogate-based optimization technique) was less successful at locating all optima than a system of agents, for an equivalent number of function evaluations, showing that exploration through search space partitioning is an important part of the multi-agent algorithm. Secondly, in the 2D example it was found that our multi-agent optimization approach benefits from a reduced computational budget devoted to random search: at constant number of evaluations, agents with an initially large DOE are less efficient at locating optimal feasible solutions than agents with smaller DOEs (hence with more iterations). In the 6D example, this remark is true looking at the median cost function evolution.

We think that the proposed agent optimization method has a great potential for parallel computing. Indeed, as the number of computing nodes n increases, the calculation of the expensive objective and constraints functions scales with $1/n$ in terms of wall-clock time. But the speed at which problems can be solved becomes then limited by the time taken by the optimizer, i.e., the process of generating a new candidate solution. In the algorithm we have developed, the optimization task itself can be divided among the n nodes through agents. We plan to explore how agents can provide a useful paradigm for optimizing in parallel, distributed, asynchronous computing environments.

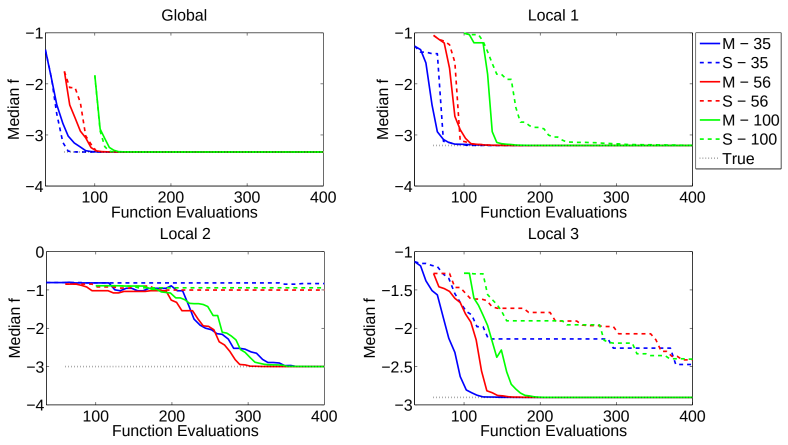


Fig. 10. Median objective function value of solution closest to each optimum with number of function evaluations

References

1. F. Aurenhammer. Voronoi diagrams: a survey of a fundamental geometric data structure. *ACM Computing Surveys (CSUR)*, 23(3):345–405, 1991.
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. L. Dixon and G. Szego. *Towards Global Optimisation 2*, chapter The Global Optimisation Problem: An Introduction. North-Holland Publishing Company, New York, 1978.
4. B. Glaz, T. Goel, L. Liu, P. Friedmann, and R. T. Haftka. Multiple-surrogate approach to helicopter rotor blade vibration reduction. *AIAA Journal*, 47(1):271–282, 2009.
5. J. Hartigan and M. Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979.

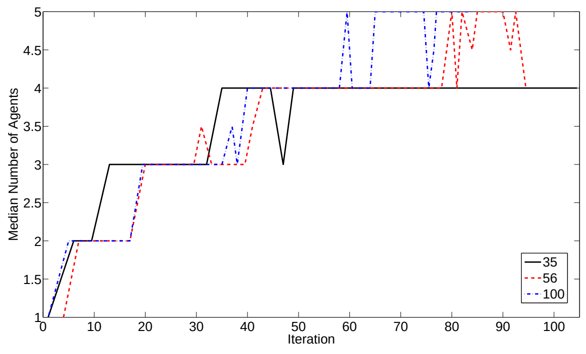


Fig. 11. For a multi-agent system, the median number of agents

6. J. Holmgren, J. A. Persson, and P. Davidsson. Agent based decomposition of optimization problems. In *First International Workshop on Optimization in Multi-Agent Systems*, 2008.
7. R. Jin, X. Du, and W. Chen. The use of metamodeling techniques for optimization under uncertainty. *Structural and Multidisciplinary Optimization*, 25(2):99–116, 2003.
8. J. Kleijen. *Design and analysis of simulation experiments*. Springer, 2008.
9. J. P. Li, M. E. Balazas, G. Parks, and P. J. Clarkson. A species conserving genetic algorithm for multimodal function optimization. *Evolutionary Computation*, 10(3):207–234, 2002.
10. 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.
11. MATLAB. *version 7.9.0.529 (R2009b)*, chapter fmincon. The MathWorks Inc., Natick, Massachusetts, 2009.
12. P. J. Modi, W. Shen, M. Tambe, and M. Yokoo. ADOPT: Asynchronous Distributed Constraint Optimization with Quality Guarantees. *Artificial Intelligence*, 161(2):149–180, 2005.
13. K. E. Parsopoulos and M. N. Vrahatis. *Artificial Neural Networks and Genetic Algorithms*, chapter Modification ofthe Particle Swarm Optimizer for Locating All the Global Minima, pages 324–327. Springer, 2001.
14. N. V. Queipo, R. T. Haftka, W. Shyy, and T. Goel. Surrogate-based analysis and optimization. *Progress in Aerospace Sciences*, 41:1–28, 2005.
15. P. J. Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Computational and Applied Mathematics*, 20:53–65, 1987.
16. J. Sacks, W. J. Welch, M. T. J., and H. P. Wynn. Design and analysis of computer experiments. *Statistical Science*, 4(4):409–435, 1989.
17. A. Samad, K. Kim, T. Goel, R. T. Haftka, and W. Shyy. Multiple surrogate modeling for axial compressor blade shape optimization. *Journal of Propulsion and Power*, 25(2):302–310, 2008.
18. M. Sasena, P. Papalambros, and P. Goovaerts. Global optimization of problems with disconnected feasible regions via surrogate modeling. In *9th AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, Atlanta, GA, 2002.
19. Y. Shoham and K. Leyton-Brown. *Multiagent Systems: Algorithmic, Game-Theoric, and Logical Foundations*. Cambridge University Press, 2009.
20. T. W. Simpson, J. D. Peplinski, P. N. Koch, and J. K. Allen. Metamodels for computer based engineering design: survey and recommendations. *Engineering with Computers*, 17(2):129–150, 2001.
21. A. Torn and A. Zilinskas. Global optimization. In *Lecture Notes in Computer Science 350*. Springer Verlag, 1989.
22. F. A. C. Viana. *Multiple Surrogates for Prediction and Optimization*. PhD thesis, University of Florida, 2011.
23. F. A. C. Viana and R. T. Haftka. Using multiple surrogates for metamodeling. In *7th ASMO-UK/ISSMO International Conference on Engineering Design Optimization*, 2008.
24. I. Voutchkov and A. J. Keane. Multiobjective optimization using surrogates. In *7th International Conference on Adaptive Computing in Design and Manufacture*, pages 167–175, Bristol, UK, 2006.
25. G. G. Wang and T. W. Simpson. Fuzzy clustering based hierarchical metamodeling for design space reduction and optimization. *Engineering Optimization*, 36(3):313–335, 2004.
26. D. Zhao and D. Xue. A multi-surrogate approximation method for metamodeling. *Engineering with Computers*, 27:139–153, 2005.