# Dynamic Partitioning for Balancing Exploitation and Exploration in Constrained Optimization: A Multi-Agent Approach

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 Superiéure des Mines de Saint-Étienne, Saint-Étienne, France*

[3] *CNRS UMR 6158*

**In this article, we define optimization subtasks that employ different approximations of the data in subregions through the choice of surrogate, which creates surrogate-based agents. Through design space partitioning, which assigns agents to subregions of the design space, the agents solve the optimization problem in their respective subregions, and use the feasibility and objective function value to assess the value of the solutions in order to center the subregions at local and global optima. Further, we introduce methods to create agents at run-time which allows additional exploration by creating a finer partition of the design space. The end result of this dynamic partitioning is a multi-agent system that inherently balances exploitation and exploration in the design space. We illustrate this approach on a constrained optimization problem with small, disconnected feasible regions. It was observed that the agents were effective at locating global and local optima.**

## Nomenclature

| | | |
|---|---|---|
| $c$ | = | center |
| $f$ | = | objective function |
| $\mathbb{F}$ | = | objective function values associated with design of experiments in database |
| $g$ | = | constraint |
| $\mathbb{G}$ | = | constraint values associated with design of experiments in database |
| $t$ | = | time |
| $x$ | = | design variables |
| $\mathbb{X}$ | = | design of experiments in database |
| $\hat{f}$ | = | surrogate prediction of $f$ |
| $\hat{g}$ | = | surrogate prediction of $g$ |

## I. Introduction

Complex optimization problems often require repeated computationally expensive simulations, such as those found in high fidelity finite element analyses. Therefore, many researchers in the field have focused on the development of optimization methods adapted to expensive simulations. The main ideas underlying

*Graduate Research Assistant, Mechanical and Aerospace Engineering/Institut Henri Fayol, AIAA student member

†CNRS permanent research associate, Institut Henri Fayol

‡Associate Professor, Institut Henri Fayol

§Distinguished Professor, Mechanical and Aerospace Engineering, AIAA Fellow such methods are i) the use of surrogates ii) problem decomposition iii) problem approximation iv) parallel computation.

In computer science, multi-agent systems solve complex problems by decomposing them into autonomous sub-tasks. A general definition posits a multi-agent system to be comprised of several autonomous agents with possibly different objectives that work toward a common task. In terms of optimization, a multi-agent system could solve decomposed problems such that the agents only know subproblems.

In a past study,[1] we described a methodology to approximate complex, computationally expensive reliability-based design optimization problems, and use the approximated problems to efficiently find a solution with a high level of accuracy. The process took advantage of multi-agent techniques. Each agent built a different surrogate and used it to find an optimum. In effect, this process creates "surrogate-based optimization agents". Each agent defined a different search process. Through cooperation (exchange of search points), the agents system defined high-level methodologies whose efficiency and robustness were studied.

The use of surrogates to replace expensive simulations and experiments in optimization has been well documented.[2–5] Because several surrogates can be inferred from a finite set of expensive simulations, some authors have proposed strategies for using multiple surrogates for optimization.[6–9] Surrogate-based agents are related to multi-surrogates, but the reference to "agents" stresses that agents can change strategies at runtime and their actions are asynchronous. There are several strategies that agents may employ:

1. Agents solve the same problem with different strategies.[1]
2. Agents solve parts of the complete problem. In a previous study,[10] we partitioned the design space such that each agent has to search in a smaller subregion.

In all cases, the agents' individual strategies are completed by a coordination mechanism, which, in the context of surrogate-based optimization, will consist of a policy for sharing the results of high fidelity simulations and partitioning the design space accordingly. In addition, agents can be deleted if they are not found to be useful and, vice versa, new agents can be created. Creating an agent at a region of the design space is a way to explore the design space, and is a point of emphasis in this article.

As stated above, we have examined the partitioning of a design space to define an agent's subregion.[10] There are several existing methods of fitting local surrogates to subregions of the design space that are partitioned using clustering techniques. Zhao and Xue[11] presented a methodology to divide the design space into clusters using the Gaussian mixture model. The Gaussian mixture model assigned points to clusters by maximizing membership probabilities. Local surrogates were fit to each cluster, and a global surrogate was fit to the entire space, merging the local surrogates by Bayes' law. Wang and Simpson[12] used response surfaces, kriging models, and points sampled from these surrogates to reduce the design space to only interesting regions. The surrogates were used to generate inexpensive points, which were then clustered using fuzzy c-means clustering. The purpose of the clustering was to identify a region with low-function values for further high-accuracy approximation and discard regions with high function values.

For our multi-agent method, the design space partitioning allows the agents to stabilize around local optima, with the goal of locating local (therefore also global) optima. For real-world problems, the ability to locate many optima is desirable as the global optimum may be too expensive to find, and because it provides the designer with a diverse set of acceptable designs. Others have developed methods to locate many local optima, mostly based on particle swarm[13–15] or genetic algorithms.[16]

The majority of the aforementioned research focused on unconstrained optimization problems or box constrained problems, whereas the multi-agent methods that have we have developed can handle both unconstrained and constrained optimization problems. In constrained surrogate-based optimization, Sasena et al.[17]extended the efficient global optimization (EGO)[18] algorithm such that the sampling criterion considered the probability of feasibility of a point as a complement to the expected improvement on the present best solution. The method was shown to be efficient at solving problems where feasible regions of the design space were small or disconnected. Other approaches to solving constrained optimization problems include a two-phase process in which the first phase finds a feasible solution and the second phase solves a bi-objective problem where the original objective function and measure of constraint violation were the objectives. This was a method put into place in genetic algorithms by Venkatraman and Yen.[19] This was a development in an approach to constraint handling by a bi-objective formulation used earlier by Fletcher and Leyffer,[20] that has also been applied to genetic algorithms and to particle swarm optimization.[21–23]

In this article, we will demonstrate how the agents have the ability to 1) handle constraints, 2) use the creation of agents for exploration, 3) exploit and explore within the subregions to find optima, and 4) locate and stabilize around global and local optima. This ability to locate and stabilize around local optima is due to the exploration that is done by the agents themselves and through the creation of agents. In addition, we describe how agents use the feasibility and objective function value of a point to assess the value of the point. In the next section, we provide a background on surrogate-based agents. We then describe the methods of design space partitioning and simulation point allocation that are intended to distribute local optima among the partitions while maximizing the accuracy of the surrogate. Next, the methods of agent creation and deletion are introduced. A constrained optimization example is provided to illustrate the multi-agent approach.

## II. Definition of Agents for Surrogate-Based Optimization

Let us consider the general formulation of a constrained optimization problem shown in Eq.(1).

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

In surrogate-based optimization, a surrogate is built from a design of experiments (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 of this approximate problem is denoted $\hat{x}^*$.

Because of the approximation, $\hat{x}^*$ may not be sufficiently close to the nearest local optimum of the true function, so more iterations may be needed to find the true optimum, with iteration index denoted by the time $t$. That is, after the optimum of the problem in Eq.(2) is found, the true values of $f$ and $g$ are calculated and included in the DOE along with $\hat{x}^*$. In general, a completely new DOE may be constructed around $\hat{x}^*$. However, in the present treatment only $\hat{x}^*$ is added to the updated DOE. In 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. This surrogate-based optimization procedure is summarized in Algorithm 1. Note that in the algorithm

**Algorithm 1** Procedure for surrogate-based design optimization

```
1: t = 1 (initial state)
2: while $t \leq t^{max}$ do
3:     Build surrogates $\hat{f}$ and $\hat{g}$ from $(\mathbb{X}^t, \mathbb{F}^t, \mathbb{G}^t)$
4:     Global optimization $\rightarrow \hat{x}^*$ (Here, solve: $\underset{x \in \mathcal{S}}{\text{minimize}}\ \hat{f}(x)$ subject to $\hat{g}(x) \leq 0 \rightarrow \hat{x}^*$
5:             if $\hat{x}^*$ is near $\mathbb{X}^t$ or infeasible, solve $\underset{x \in \mathcal{S}}{\text{maximize}}$ distance$(\mathbb{X}^t) \rightarrow \hat{x}^*$)
6:     Calculate $f(\hat{x}^*)$ and $g(\hat{x}^*)$
7:     Update database $(\mathbb{X}^{t+1}, \mathbb{F}^{t+1}, \mathbb{G}^{t+1}) \cup (\hat{x}^*, f(\hat{x}^*), g(\hat{x}^*))$
8:     $t = t + 1$
9: end while
```

the alternating surrogate optimization and space filling together define an infill sampling criterion to find $\hat{x}^*$. In global surrogate-based optimization, the infill sampling criterion can be based on, for example, expected improvement or probability of improvement on the present best solution. Such criteria consider the uncertainty structure of the surrogate, along with the prediction provided by the surrogate to explore the design space. In this paper, the surrogate is used to solve the actual optimization problem shown in Eq.(2) using only the surrogate prediction. This strategy would likely be ineffective with a single surrogate over the entire design space as it could lead to convergence at a single point that is possibly away from the global optimum, which is the reason why it needs to be completed with space filling phases.

Algorithm 1 can be thought of as the procedure followed by a single surrogate-based agent in one iteration. In addition, when there is more than one agent, 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 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 $i$th agent. Each agent must consider only the points in its subregion, which are available in its internal database $(\mathbb{X}^t, \mathbb{F}^t, \mathbb{G}^t)_{internal}$. The subregion of an agent is defined by the position of its center $c$. A point in the design space belongs to the subregion with the nearest center, where the distance is the Euclidean distance. This creates subregions that are Voronoi cells.[24] The choice of where to place the center is the subject of the next section. Note, however, that the term "center" does not imply that it is at the geometric center of the subregion, and in principle it can be very close to its boundary.

Figure 1 illustrates the partition of a two-dimensional design space into four subregions for four agents, which requires four centers. In this example, we place the centers randomly. The points in the design space that correspond to each subregion are represented by the four different colors.

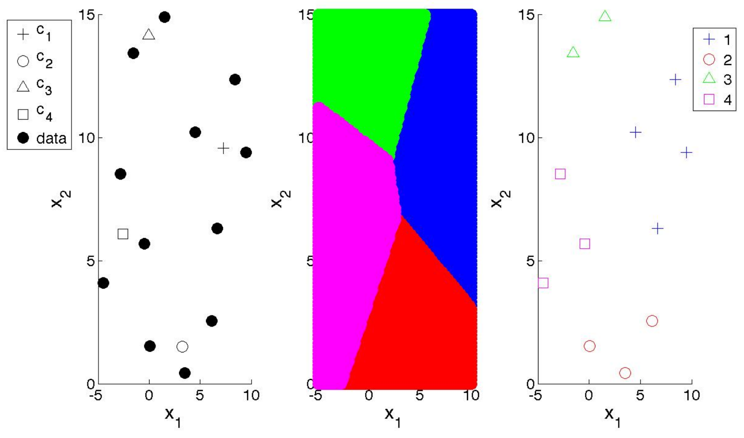


**Figure 1. Two-dimensional partitioning example showing data points and four centers (left), corresponding to four agents and subregions (middle), and assignment of data points to each agent and subregion (right).**

Once the subregions are defined, each agent fits several surrogates and chooses the one that maximizes the accuracy in its subregion. 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. 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 refitting the surrogate and leaving out a point, and measuring the error at that point. This is done 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, the optimization is performed to solve the problem in Eq.(2) inside the subregion. 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 a point that is too close to another agent, the agent then explores in the subregion. The definition of "too close" to another agent is a choice of the user. Since the optimization is performed using the surrogate predictions, it is relatively cheap to start the optimization at multiple initial points. The solution with the minimum objective function value that satisfies all constraints at a distance far enough from an existing point is the solution. This allows the agent to locate multiple optima in subregion. If no solution from the multiple start satisfies these criteria, the agent explores. To explore, the agent adds a point to the database that maximizes the minimum distance from the points already in its internal database. The true values $f$ and $g$ of the iterate are then calculated, and $(\hat{x}^*, f(\hat{x}^*), g(\hat{x}^*))$ is added to the global database. The next iteration then starts by repartitioning the design space using the points in the global database. The procedure of a single agent is given in Algorithm 2.

**Algorithm 2** Agent $i$ optimization in its subregion. The method of partitioning is described in Algorithm 4

```
1:  t = 1 (initial state)
2:  while $t \leq t^{max}$ do
3:     Partition the design space (Algorithm 4), get $c_i$
4:     Define the local design space as $\mathcal{P}_i = \{x \in \mathcal{S}$ s.t.$||x - c_i||^2 \leq ||x - c_j||^2 \; , \; j \neq i\}$
5:     Update internal database from the new design space partition
6:     Check for deletion and/or creation of agents (To be discussed in Sec.V)
7:     Build surrogates $\hat{f}$ and $\hat{g}$ from internal database $(\mathbb{X}^t, \mathbb{F}^t, \mathbb{G}^t)_{internal}$
8:     if Not sufficient number of points in internal database to build a surrogate then
9:        Get points from other agents closest to $c_i$
10:       Build surrogates
11:    end if
12:    Choose best surrogate based on partial $PRESS_{RMS}$ error
13:    Global optimization → $\hat{x}^*$ (Here, use multiple starts to solve: $\underset{x \in \mathcal{P}_i}{\text{minimize}} \; \hat{f}(x)$ subject to $\hat{g}(x) \leq 0 \rightarrow \hat{x}^*$
14:          if $\hat{x}^*$ is near $\mathbb{X}^t$ or infeasible, solve $\underset{x \in \mathcal{P}_i}{\text{maximize}} \; \text{distance}(\mathbb{X}^t) \rightarrow \hat{x}^*$)
15:    Calculate $f(\hat{x}^*)$ and $g(\hat{x}^*)$
16:    Update global database $(\mathbb{X}^{t+1}, \mathbb{F}^{t+1}, \mathbb{G}^{t+1})_{global} \cup (\hat{x}^*, f(\hat{x}^*), g(\hat{x}^*))$
17:    $t = t + 1$
18: end while
```

## III. Combining Feasibility and Objective Function Values to Assess the Value of a Point

In the multi-agent method, the comparison between two points is important, particularly in partitioning the design space into subregions for the agents that we will describe in the following section. In this section, we describe how to compare two points, considering the feasbility and objective function value, to decide which is the better of the two points. 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" $x_n$. The conditions to determine the better of two points are given in Algorithm 3.

The algorithm may be summarized as follows: If both points are feasible, then the point with the lower objective function value is the better point. If both points are infeasible, the better of the two points must always have the smaller maximum violation of the constraint functions (i.e., closer to the feasible state). If one point is feasible whereas the other is infeasible, the better point is the feasible point.

## IV. Design Space Partitioning

The method of design 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 also explore the subregion around the local optimum.

For the initial partitioning, in which there is no information on local optima, we follow a different logic: the initial partition is made by k-means clustering.[25] K-means clustering is a method that partitions $n$ data points into $k$ clusters such that each observation belongs to the cluster with the nearest mean. After the points are clustered in the first iteration ($t = 1$), agents are assigned to the partitioned subspaces and the points in the subspaces define the internal database $(\mathbb{X}^t, \mathbb{F}^t, \mathbb{G}^t)_{internal}$ of each agent. The process then follows the algorithm in Algorithm 2 until the end of the first iteration.

From the second iteration on, the center of the subregion is moved to the "best" point in the subregion in terms of feasibility and objective function value. This is done by comparing the center at the last iteration $c^{t-1}$ to the last point added by the agent $\hat{x}^{*t-1}$ as described in Sec.III (for two centers, instead of points $x$

**Algorithm 3** Algorithm to determining if, for two points $x_m$ and $x_n$, $x_m$ "is better than" $x_n$ ($x_m \succ x_n$) and vice versa. Note that for the algorithm below, the time (superscript $t$) is omitted as the algorithm is valid for the comparison of any two points at any time.

```
1: Given $f(x_m), f(x_n), max(g(x_m)), max(g(x_n))$
2: if $max(g(x_m)) \leq 0$ & $max(g(x_n)) \leq 0$ then
3:     if $f(x_m) \leq f(x_n)$ then
4:         $x_m \succ x_n$
5:     else
6:         $x_n \succ x_m$
7:     end if
8: else if $max(g(x_m)) \leq 0$ & $max(g(x_n)) > 0$ then
9:     $x_m \succ x_n$
10: else if $max(g(x_m)) > 0$ & $max(g(x_n)) \leq 0$ then
11:     $x_n \succ x_m$
12: else if $max(g(x_m)) > 0$ & $max(g(x_n)) > 0$ then
13:     if $max(g(x_m)) \leq max(g(x_n))$ then
14:         $x_m \succ x_n$
15:     else
16:         $x_n \succ x_m$
17:     end if
18: end if
```

we would consider the centers $c$). 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. The conditions in which the center is moved are summarized in Algorithm 4, where after partitioning, the solution process continues with Algorithm 2.

**Algorithm 4** Partitioning design space (calculate $c_i$'s) around local optimum in subregion $i$. The iteration is represented by $t$.

```
1: if $t = 1$ (initial state) then
2:     Use k-means clustering to find centers to partition design space $\rightarrow c_i^t$
3: else
4:     if $c_i^{t-1} \succ \hat{x}_i^{*t-1}$ then
5:         $c_i^t = c_i^{t-1}$
6:     else
7:         $c_i^t = \hat{x}_i^{*t-1}$
8:     end if
9: end if
```

## V. Creation and Deletion of Agents

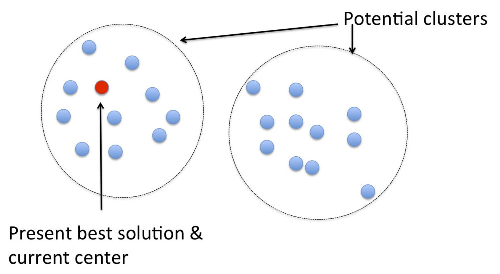
In this section, we introduce methods to create and delete agents. Deletion of agents prevents agents from crowding the same area, allowing one agent to capture the behavior in a region. Creating an agent is a way to explore the design space as it refines the partitioning of the design space in addition to the search that each agent can perform in its subregion. The deletion and creation of agents occurs at the beginning of each iteration: agents are first deleted (if necessary), the points belonging to the deleted agent(s) are distributed to the remaining agents based on distance from the center of the remaining agents' subregions, and then each remaining agents are examined to determine whether any new agent can be created given the points in the subregions.

### A. Deletion

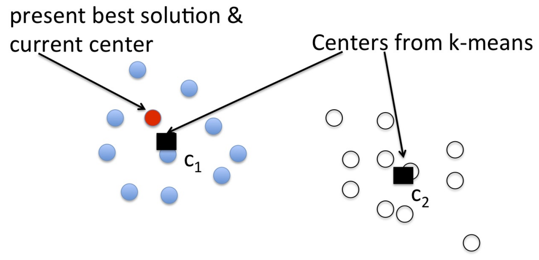
Agents are deleted if the centers of the agents' subregions are too close as measured by the Euclidean distance between the centers. We measure the minimum Euclidean distance between two centers as a percentage of the maximum possible Euclidean distance between points in the design space. When examining the agents, the agent with the center with the lowest performance is deleted. For example, if agents 1 and 2 are too close to each other and if $c_1 \succ c_2$, then agent 2 is deleted.

## B. Creation

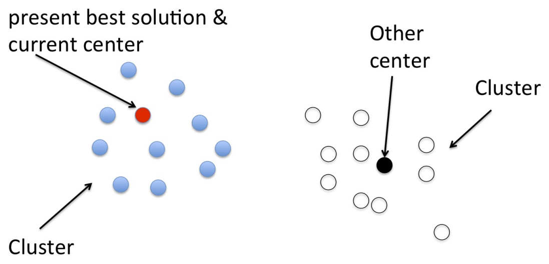
It is desirable to create an agent if it is found that points are clustered in two separate areas of a single agent's subregion, as illustrated in Fig.2(a). Such a situation can occur if there are two optima in a subregion.



(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

**Figure 2. Illustration of process used to create an agent given points in a single agent's subregion. Note: The silhouette value is checked to determine if the clustering is accepted.**

Agents are created by using k-means clustering 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 the expensive functions, which would be necessary as the centers of the subregions are compared to the next point added to decide if the center should be moved. Firstly, we measure the distance of the centers from k-means to the present best solution, and move 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,[26] validates the number of clusters by providing a measure of the within-cluster tightness and separation from other clusters for each data point. The silhouette value for each point $i$ 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.

## VI. Illustrative Example

In this section, an example is provided to illustrate the effectiveness of the methods described in the previous sections. The example is a two dimensional problem with a single constraint taken from Sasena et al.[17] 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,[27] and is referred to as *newBranin*. This problem is shown in Eq.(5).

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

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).

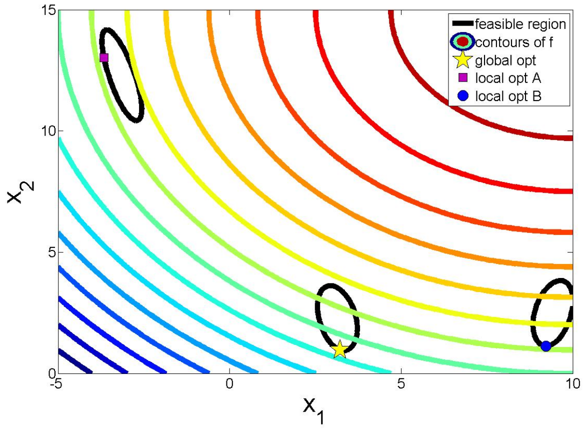


**Figure 3. Contour plot of $f$ for the *newBranin* problem. The feasible region boundaries are shown as black lines and the global optimum is shown with a yellow star at $x_1 = 3.2143$ and $x_2 = 0.9633$. Local optimum A is located at (9.2153,1.1240) and local optimum B is located at (-3.6685,13.0299).**

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. Recall that if there are not enough points in an agent's subregion to fit the surrogate, the agent will borrow the closest points from other subregions.

For all results, the parameters in Table 2 were used to solve the problem. These parameters include the minimum and maximum number of agents, parameters that dictate how close points and centers can be, and parameters that define if a a new agent should be created. The minimum distance between points is only violated if violated in every dimension.

The success and efficiency of agents was compared for three cases:

1. Single agent - Equivalent to using a single surrogate to fit the entire design space. A minimum distance of 1e-3 between points was the only parameter from Table 2 that was used, as the others are unnecessary in the single agent case. The initial size of the DOE was 12.

2. Varying the initial number of agents from 2 to 6. The initial size of the DOE was 12.

**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 quadratic response surface |
| 6 | Kriging (constant trend) | |

**Table 2. Agent Parameters**

| Parameter | Value |
|---|---|
| Max # of agents | 6 |
| Min # of agents | 2 |
| Min distance between agent centers | 10% of max possible distance in design space |
| Min distance between points | 1e-3 (absolute for each dimension) |
| Min average silhouette | 0.75 |
| Min # of points in each agent after creation | 4 |

3. The initial size of the DOE was changed (initial number of agents was fixed at 4).

In each case, a maximum of around 120 to 140 total function evaluations was allowed, by fixing the number of allowable iterations based on a maximum of 6 agents. 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,[28] with 10 initial starting points.

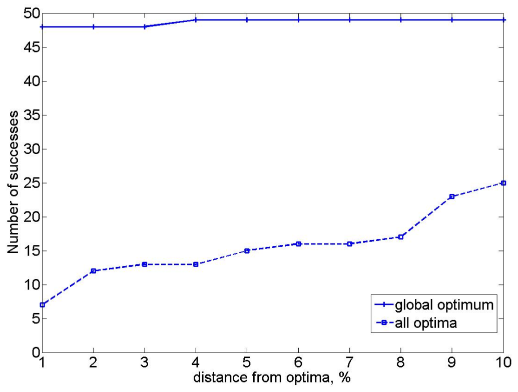
## A. Single Agent

The results of the single agent case are shown below in Fig.4. Figure 4(a) displays the number of successes in locating a feasible solution that is within some percentage distance from the optimum. This distance is the Euclidean distance between the 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 successful overall at finding a solution near the global optimum (49 successes at a 4% distance from the optimum).

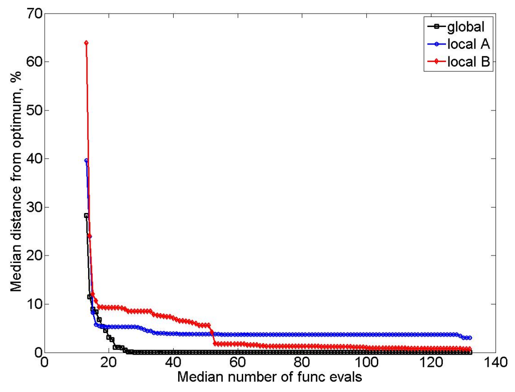
It was unsuccessful at locating solutions near to all of the optima, with only 25 successes at a distance of 10% from the optima. In general, it would be expected that the success rate in finding multiple optima would be low using a single surrogate solving only the original optimization problem in the entire design space without a means of exploration (i.e., pure exploitation). However, we observed a 50% success rate using a single agent as the agent is aided by the use of multiple starting points for the optimization and the constraint on the minimum distance between points. In addition, when no solutions from the multiple starts met these constraints, the agent explored by maximizing the minimum distance between points.

Figure 4(b) displays the median distance of the closest solution to each of the optima as a function of the median number of function evaluations. Feasibility is not considered in finding the closest solution, and the value of the constraint function $g$ at this solution is shown in Fig. 4(c). The number of function evaluations at each iteration $t$ was calculated as

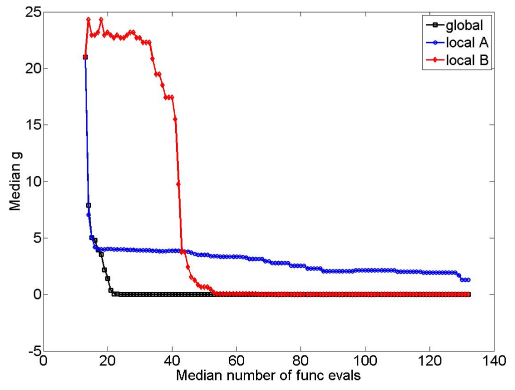
$$\text{func evaluations} = \text{ initial size of DOE} + \sum_{j=1}^{\text{t}} (\text{ \# of active agents})_j \tag{6}$$



(a) Number of Successes



(b) Distance from optima



(c) $g$ at solutions near optima

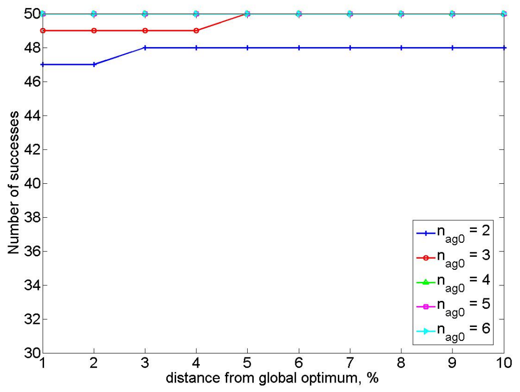
**Figure 4. For a single agent, (a) number of successes in locating optima versus distance from the optima, (b) median distance from optima versus median number of function evaluations, and (c) $g$ of solutions closest to the optima versus median number of function evaluations**

It was observed that the global optimum was found by the agent with the fewest number of function evaluations (approximately 25), followed by local optimum B. Local optimum A was the most difficult to find by the single agent, where the median $g$ shows that the closest solution was often infeasible.

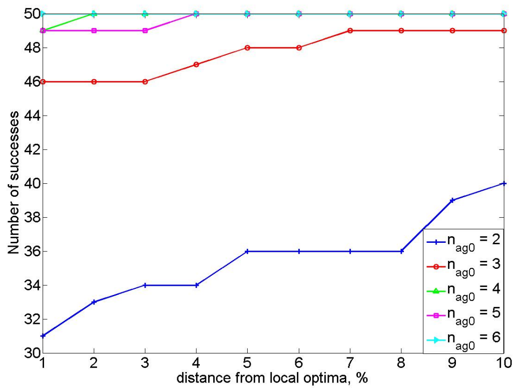
Failure to locate solutions near the optima can be attributed to a few factors: 1) the agent is drawn towards the better optimum (or optima) and adds points to refine the solution (as the surrogate is also refined as points are added), 2) exploration does not result in a point that would draw the agent close to that optima. We can control how refined the solution is around an optimum by controlling the minimum distance between points that are added. For this problem, this value is 1e-3, which is an absolute value for each dimension. If this value is reduced, this could prevent points from being repeatedly drawn to the same optima and ignite exploration. Exploration can also be increased by having a variable number of agents and allowing the creation of agents at run-time. The effect of having a variable number of agents at run-time was explored in the remaining two cases.

## B. Varying the initial number of agents

Figure 5 displays the success at locating a solution within some distance from the global optimum and all optima with the initial number of agents $n_{ag0}$ ranging from 2 to 6. It was observed that with initially 4 or more agents, the agents were successful in all 50 repetitions at locating a solution within 1% of the global optimum and within 4% of all local optima. With 2 agents initially, the agents were about as successful as a single agent at locating a solution near the global optimum, and also more successful in locating all optima.



(a) Global optimum



(b) All Optima

**Figure 5. Varying the initial number of agents, success in locating (a) the global optimum and (b) all optima at some distance from the optima**

The median number of active agents through the 20 iterations is shown in Fig. 6. With the exception of the $n_{ag0} = 2$ case, all cases converged to use 5 agents. This lack of creation for the case with 2 agents at the initial stage may explain why it was unable to locate all optima as successfully as the others.

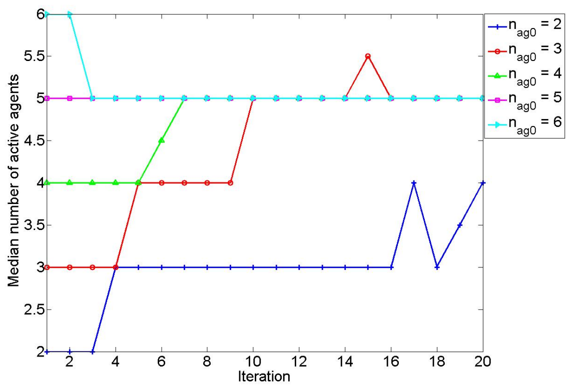
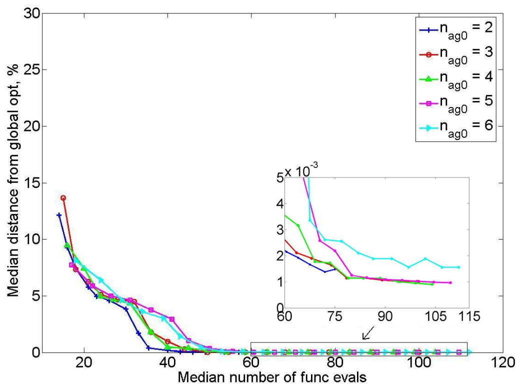


**Figure 6. Varying the initial number of agents, the median number of active agents through 20 iterations**

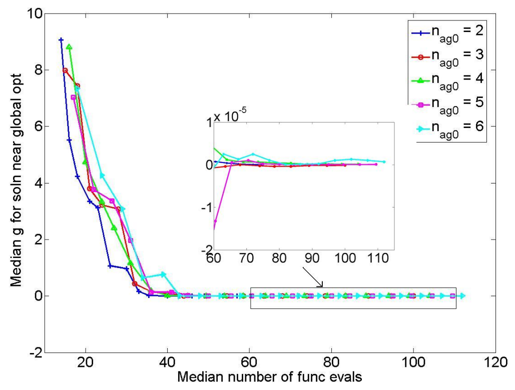
The median distance of the closest solution to each of the optima and the value of the constraint function $g$ versus the median number of function evaluations is shown in Fig. 7. Feasibility was not considered when locating the closest point. Note that there were values of the median distance to the local optima around 1% for the $n_{ag0} = 2$ case, whereas we observed only 31 successes in finding a feasible solution at this distance. However, when considering the median values, we observed success at finding a solution with a median distance less than 1% at a solution that was nearly feasible. As with the single agent case, the agents were first able to find the global optimum, followed by local optimum B, and finally local optimum A. The efficiency of the cases with 4 to 6 agents initially was nearly equal. The case with an initial DOE of 20 was generally the most efficient at converging to the local optima although mostly from the infeasible region of the design space.

## C. Effect of the size of the initial DOE

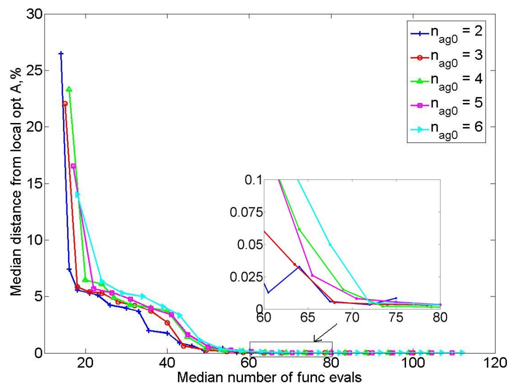
The final case examined the effect of the size of the initial DOE for $n_{nag0} = 4$. The initial DOE size was varied from 20 to 100, in increments of 10. The DOEs were found using LHS. The number of iterations for each DOE was chosen such that the maximum number of possible function evaluations was equal to that of



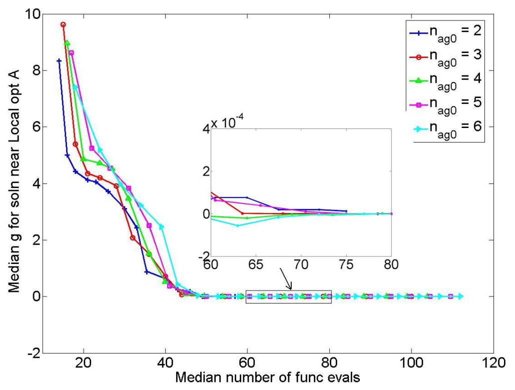
(a) Distance from global optimum



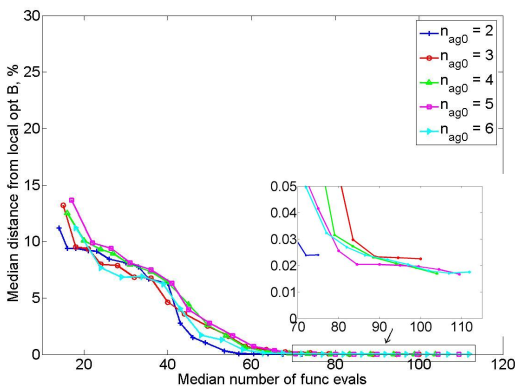
(b) $g$ of soln near global optimum



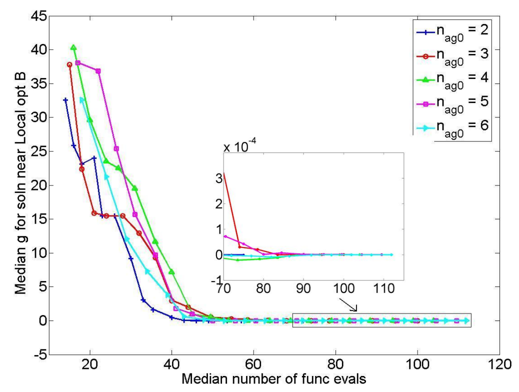
(c) Distance from local optimum A



(d) $g$ of soln near local optimum A



(e) Distance from local optimum B

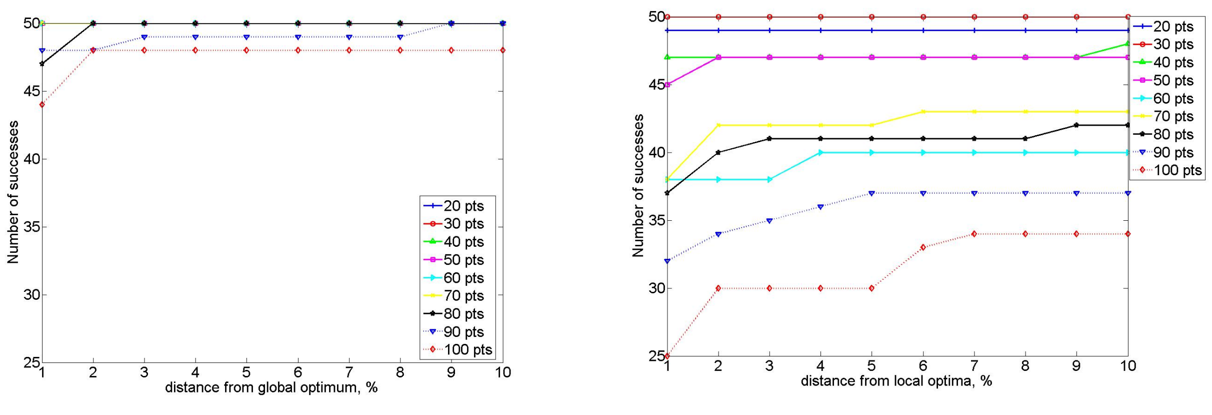


(f) $g$ of soln near local optimum B

**Figure 7. Varying the initial number of agents, distance and $g$ value of closest solution to the global optimum (top), local optimum A (middle), and local optimum B (bottom) versus median number of function evaluations**

the case with an initial DOE of 12 for 20 iterations. The maximum number of possible function evaluations was equal to max # of agents $\times$ # of iterations $+$ initial size of DOE, which for an initial DOE of 12 for 20 iterations with a maximum of 6 agents is 132 possible function evaluations.

Figure 8 displays the success at locating a solution within some distance from the global optimum and all optima. It was observed that with a DOE of greater than 30 points, the agents were unsuccessful at locating the global optimum in all repetitions. For the large initial DOEs of 90 and 100, success at finding all optima was close to 35 successes, similar to the $n_{ag0} = 2$ case with an initial DOE of 12 and 20 iterations. For the small DOEs with more agent iterations, there was more success at finding a solution near the global



(a) Global optimum (b) All Optima

**Figure 8. Varying the size of the initial DOE, success in locating a feasbile solution near (a) the global optimum and (b) all optima at some distance from the optima**

optimum, but only the cases with an initial DOE of 20 and 30 points had a large number of successes (49 and 50 successes, respectively).

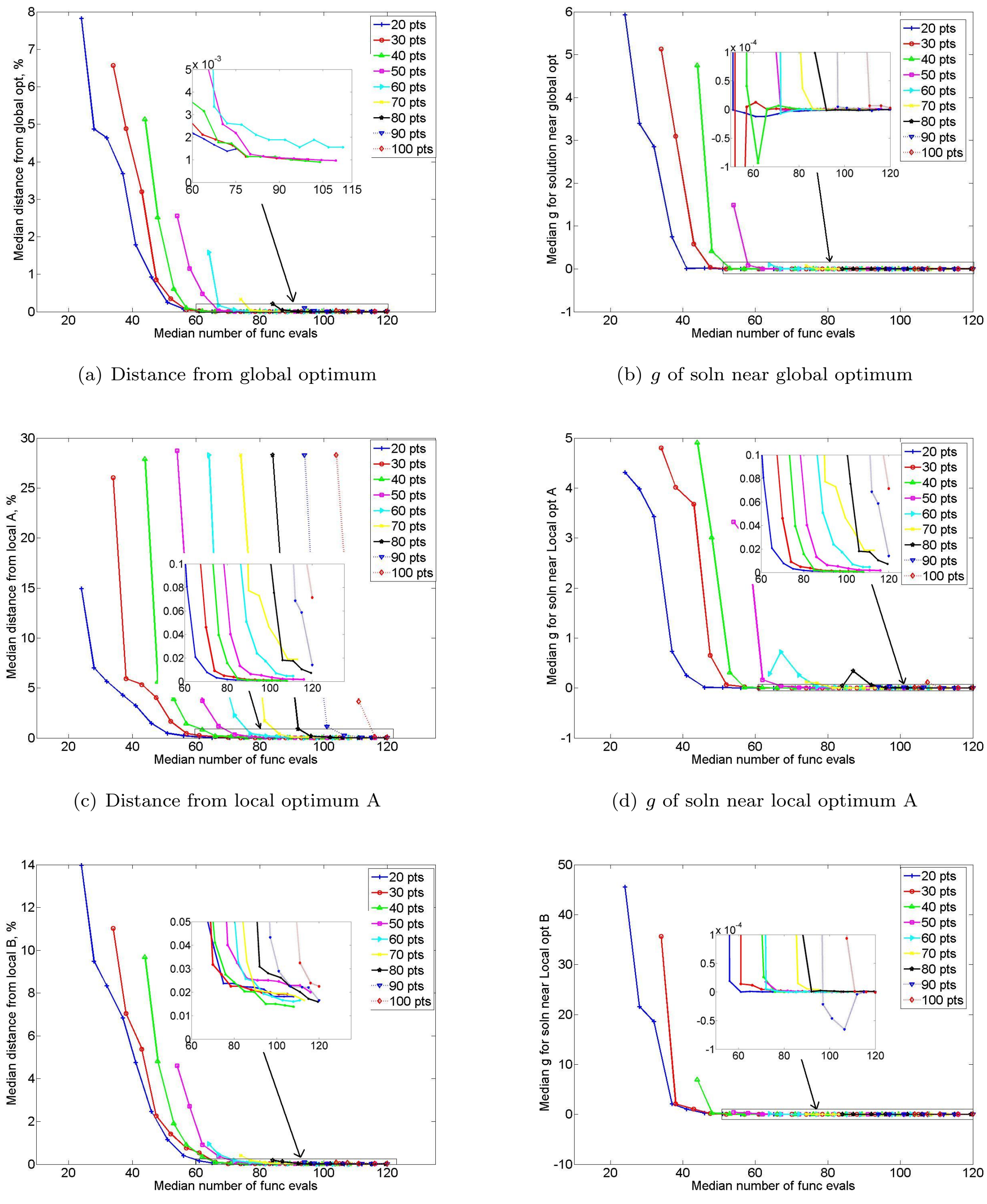
The median distance of the closest solution to each of the optima and the value of $g$ at this solution versus the median number of function evaluations is shown in Fig. 9. Again, feasibility was not considered in finding the closest solutions, so in many of these cases the closest solutions approached the optima from the infeasible region, yet we observed that the median values are at or near $g = 0$. Without considering infeasibility, the case with an initial DOE of 20 was generally the most efficient at converging to the local optima.

## VII. Conclusions and Future Work

This paper introduced an optimization method where surrogate-based agents partition the original design space and optimize therein. The centers of the agents' subregions moved to stabilize around optima, and agents were created at run-time as a means of exploration. Through this partitioning of the subregions and creation of the agents, the multi-agent system was able to balance exploitation and exploration in the design space.

The single agent was the least successful at locating all optima for a nearly equal number of function evaluations, showing that design space partitioning is an important part of the agent algorithm. To maximize the possibility of finding solutions near all optima, the initial number of agents should be large, and creation should be allowed. It was found that, for a fixed number of maximum expensive function evaluations, using agents with an initially large DOE with a fewer agent iterations is less successful and less efficient at locating feasible solutions close to the optima compared to smaller DOEs with more agent iterations. This leads us to believe that the ability of agents to learn in the design space and explore through the iterations is important in locating multiple optima.

As distributed and parallel computing becomes more commonplace, there is a need for optimization algorithms whose performance scales well with the number of computing nodes. Our agent algorithm fits distributed computing for the following reason: the objective and constraint functions are typically expensive in comparison to the optimizer. If $n$ evaluations of these objective and constraint functions can be performed simultaneously on $n$ nodes, the rate at which new evaluations arrive increases $n$ times. As $n$ increases, the speed at which problems can be solved will be then limited by the time taken by the optimizer to propose new points. In the algorithm we have developed, the optimization task itself is decomposed into agents that can be distributed on many nodes, thus relieving this bottleneck. We plan to further study how multiple asynchronous agents perform in a distributed computing context.



(a) Distance from global optimum

(b) $g$ of soln near global optimum

(c) Distance from local optimum A

(d) $g$ of soln near local optimum A

(e) Distance from local optimum B

(f) $g$ of soln near local optimum B

**Figure 9. Varying the size of the initial DOE, distance and $g$ value of closest solution to the global optimum (top), local optimum A (middle), and local optimum B (bottom) versus median number of function evaluations**

## Acknowledgments

This work has benefited from funding from Agence Nationale de la Recherche (French National Research Agency) with ANR-09-COSI-005 reference.

## References

[1]Villanueva, D., Le Riche, R., Picard, P., and Haftka, R. T., "A Multi-Agent System Approach To Reliability Based Design Optimization Including Future Tests," *12e Congrès de la Société Francaise de Recherche Opérationnelle et d'Aide à la Decision (ROADEF'11)*, Saint-Etienne, France, 2011.

[2]Jin, R., Du, X., and Chen, W., "The use of metamodeling techniques for optimization under uncertainty," *Structural and Multidisciplinary Optimization*, Vol. 25, No. 2, 2003, pp. 99–116.

[3]Queipo, N. V., Haftka, R. T., Shyy, W., and Goel, T., "Surrogate-based analysis and optimization," *Progress in Aerospace Sciences*, Vol. 41, 2005, pp. 1–28.

[4]Sacks, J., Welch, W. J., J., M. T., and Wynn, H. P., "Design and analysis of computer experiments," *Statistical Science*, Vol. 4, No. 4, 1989, pp. 409–435.

[5]Simpson, T. W., Peplinski, J. D., Koch, P. N., and Allen, J. K., "Metamodels for computer based engineering design: survey and recommendations," *Engineering with Computers*, Vol. 17, No. 2, 2001, pp. 129–150.

[6]Voutchkov, I. and Keane, A. J., "Multiobjective optimization using surrogates," *7th International Conference on Adaptive Computing in Design and Manufacture*, Bristol, UK, 2006, pp. 167–175.

[7]Samad, A., Kim, K., Goel, T., Haftka, R. T., and Shyy, W., "Multiple surrogate modeling for axial compressor blade shape optimization," *Journal of Propulsion and Power*, Vol. 25, No. 2, 2008, pp. 302–310.

[8]Viana, F. A. C. and Haftka, R. T., "Using multiple surrogates for metamodeling," *7th ASMO-UK/ISSMO International Conference on Engineering Design Optimization*, Bath, UK, 2008.

[9]Glaz, B., Goel, T., Liu, L., Friedmann, P., and Haftka, R. T., "Multiple-surrogate approach to helicopter rotor blade vibration reduction," *AIAA Journal*, Vol. 47, No. 1, 2009, pp. 271–282.

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

[11]Zhao, D. and Xue, D., "A multi-surrogate approximation method for metamodeling," *Engineering with Computers*, Vol. 27, 2005, pp. 139–153.

[12]Wang, G. G. and Simpson, T. W., "Fuzzy Clustering Based Hierarchical Metamodeling for Design Space Reduction and Optimization," *Engineering Optimization*, Vol. 36, No. 3, 2004, pp. 313–335.

[13]Brits, R., Engelbrecht, A. P., and van den Bergh, F., "Locating multiple optima using particle swarm optimization," *Applied Mathematics and Computation*, Vol. 189, No. 2, 2007, pp. 1859–1883.

[14]Parsopoulos, K. E. and Vrahatis, M. N., *Artificial Neural Networks and Genetic Algorithms*, chap. Modification ofthe Particle Swarm Optimizer for Locating All the Global Minima, Springer, 2001, pp. 324–327.

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

[16]Li, J. P., Balazas, M. E., Parks, G., and Clarkson, P. J., "A Species Conserving Genetic Algorithm for Multimodal Function Optimization," *Evolutionary Computation*, Vol. 10, No. 3, 2002, pp. 207–234.

[17]Sasena, M., Papalambros, P., and Goovaerts, P., "Global Optimization of Problems with Disconnected Feasible Regions Via Surrogate Modeling," 2002.

[18]Jones, D. R., Schonlau, M., and Welch, W. J., "Efficient Global Optimization of Expensive Black-Box Functions," *Journal of Global Optimization*, Vol. 13, No. 4, pp. 455–492.

[19]Venkatraman, S. and Yen, G., "A generic framework for constrained optimization using genetic algorithms," *IEEE Trans Evol Comput*, Vol. 9, 2005, pp. 424–435.

[20]Fletcher, R. and Leyffer, S., "Nonlinear programming without a penalty function," *Math Program*, Vol. 91, No. 2, 2002, pp. 239–269.

[21]Liu, C., "New multiobjective PSO algorithm for nonlinear constrained programming problems," *Advances in cognitive neurodynamics ICCN 2007*, Springer, New York, 2007, pp. 955–962.

[22]Zhou, Y., Li, Y., He, J., and Kang, L., "Multi-objective and MGG evolutionary algorithm for constrained optimization," *The 2003 Congress on Evolutionary Computation*, Canberra, 2003, pp. 1–5.

[23]Venter, G. and Haftka, R., "Constrained particle swarm optimization using a bi-objective formulation," *Structural and Multidisciplinary Optimization*, Vol. 40, 2005, pp. 64–76.

[24]Aurenhammer, F., "Voronoi diagrams: a survey of a fundamental geometric data structure," *ACM Computing Surveys (CSUR)*, Vol. 23, No. 3, 1991, pp. 345–405.

[25]Hartigan, J. and Wong, M., "Algorithm AS 136: A K-Means Clustering Algorithm," *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, Vol. 28, No. 1, 1979, pp. 100–108.

[26]Rousseeuw, P. J., "Silhouettes: a Graphical Aid to the Interpretation and Validation of Cluster Analysis," *Computational and Applied Mathematics*, Vol. 20, 1987, pp. 53–65.

[27]Dixon, L. and Szego, G., *Towards Global Optimisation 2*, chap. The Global Optimisation Problem: An Introduction, North-Holland Publishing Company, New York, 1978.

[28]MATLAB, *version 7.9.0.529 (R2009b)*, chap. fmincon, The MathWorks Inc., Natick, Massachusetts, 2009.