10th World Congress on Structural and Multidisciplinary Optimization
May 19 - 24, 2013, Orlando, Florida, USA

Locating Multiple Candidate Designs with Surrogate-Based Optimization

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

[1]University of Florida, Gainesville, FL, 32611, USA (dvillanu@gmail.com,haftka@ufl.edu)
[2]École Nationale Supérieure des Mines de Saint-Étienne, Saint-Étienne, France ({leriche,picard}@emse.fr)
[3] CNRS LIMOS UMR 6158

1. Abstract

Increases in computational power have led to a growing interest in finding global rather than local optima. However, in the design of complex systems, there is substantial risk that optima found by simulation-based optimization could prove to be useless or inferior because of modeling errors or overlooked objectives or constraints. Our recent work has established that it may be possible to develop global optimization algorithms that systematically obtain all potentially useful local optima without much increase in computational cost. This can be viewed as providing the designer with some insurance against the unexpected. In this paper, we consider different approaches to locate optima that all have in common the use of surrogate predictions to reduce the number of expensive functions. A method that we previously developed that dynamically partitions the design space for optimization with "surrogate-based" agents is compared to optimization with a global surrogate that adds multiple points at a time and the Efficient Global Optimization algorithm. It was observed that existing global optimization algorithms have potential to be adapted to locate multiple candidate designs, but the key to efficiency lies in parallelization of optimization processes.

2. Keywords: Surrogates, Multiple Optima, Partitioning

3. Introduction

In optimization courses, students are often told that defining an optimization problem properly is the most important step for obtaining a good design. However, even experienced hands often overlook important objective functions and constraints. There are also epistemic uncertainties, such as modeling errors, in the objective functions and constraint definitions that will typically perturb their relative values throughout the design space. For both reasons, alternative local optima may be better practical solutions to a given optimization problem than a single idealized global optimum. This paper will address this problem by exploring methodology to produce multiple local optima, thus providing some insurance against a discovery late in the design process that the supposed optimum design is seriously flawed because of missed or poorly modeled constraint or objective.

Advances in computer power have made it possible to move from settling on local optima to finding the global optimum, when designing engineering systems. This usually requires search in multiple regions of design space, expending most of the computation needed to define multiple alternate designs. Thus, focusing solely on locating the best design may be wasteful.

Locating multiple optima is often done with nature-inspired algorithms using niching methods (Beasley et al., 1993 [1]; Hocaoglu and Anderson, 1997 [2]). For example, Nagendra et al. [3] used a genetic algorithm to find several structural designs with comparable weight and identical load carrying capacity. However, when three of these designs were built and tested, their load carrying capacity was found to differ by 10%. Parsopoulos and Vrahas used particle swarm optimization [4]. Restarted local optimization methods with clustering (Törn and Zilinkas [5], Törn and Viitanen [6]) have also been proposed for finding many local optima. These methods require a large number of function evaluations, which is prohibitively costly if the functions are expensive to evaluate.

To reduce the cost of optimization, surrogate models are often used (e.g. Jones et al., 1998 [7], Alexandrov et al., 1998 [8]) to approximate the output of the simulations. A surrogate (or metamodel) is an algebraic expression fit to a number of simulations. Traditionally, the locations where simulations are carried out were selected independently of the optimization, so that a surrogate fitting phase preceded the optimization phase. More recently, global optimization algorithms that combine surrogate fitting and optimization have gained popularity, most notably the Efficient Global Optimization (EGO) algorithm (Schonlau, 1997 [9], Jones et al., 1998 [7]). These use adaptive sequential sampling with points added at locations with high potential of improving the design.

Our previous research has sought to extend adaptive sampling surrogate techniques to locate multiple optima. Villanueva et al., 2012 [10] proposed an approach based on the conjunction of two principles to identify many

candidate optima: i) dynamic partitioning of the search space and ii) local surrogate approximations. In this paper, we examine the effectiveness of this approach in locating multiple optima along with two methods: multiple starting points for local optimization and EGO, which is perhaps the currently favored surrogate-based global optimization algorithm. Additionally, we compare the use of global surrogate approximations in our previously developed approach in place of local surrogates.

The next section of this paper describes the general idea behind surrogate-based optimization and details two approaches, multiple starting points of multiple local optimizations and the EGO algorithm, and describes why they are interesting for this study. In Sec. 5, we present the method developed by the authors to partition the design space for optimization to identify multiple candidate designs . Section 6 compares these methods on two two-dimensional numerical examples, minimization of the Branin-Hoo and Sasena functions.

4. 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 [11]. The set of initial candidate solutions, or points, used to fit the surrogate is called the design of experiments (DOE). Well-known examples of surrogates are polynomial response surface, splines, neural networks, and kriging.

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

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

In surrogate-based optimization, a surrogate is built from a 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

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

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.

Algorithm 1: Overall surrogate-based optimization

```
t = 1 (initial state)
while t ≤ t^max do
    Build surrogates f̂ and ĝ from (𝕏^t, 𝔽^t, 𝔾^t)
    Optimization to find x̂*
    Calculate f(x̂*) and g(x̂*)
    Update database (𝕏^t, 𝔽^t, 𝔾^t) ∪ (x̂*, f(x̂*), g(x̂*))
    t = t + 1
```

The choice of how to use the surrogate prediction to find the optimum x^* is considered in the next two sub-sections. There is the choice of simply solving original optimization problem (i.e, minimizing $\hat{f}$) by solving Eq.(2), but there are many popular in-fill sampling criteria that have been developed. The following two sub-sections expand on two options to find the optimum x^* .

4.1 Multiple-Starting Points

In this research, we focus on local, gradient-based algorithms that remain efficient in high dimensions, but have the disadvantage of being highly dependent on the starting point. The choice of the starting point or "initial guess" can

affect the ability of a local algorithm to find the global optimum. Note that stochastic methods are also affected by the initial configurations (e.g., the initial population in genetic algorithms). An optimizer that uses gradients may not find the global optimum if its starting point is in the basin of attraction of a poorer local optimum. To overcome this, multiple starting points are often used, in order to take multiple trajectories to find a solution. Thus, the use of multiple starting points is good practice when using such algorithms. There is the possibility of obtaining as many solutions as starting points, though not all solutions may be unique as some starts may find the same solution. The choice of starting points is important, and typically comes from sampling methods (e.g., random sampling, grid sampling, Latin Hypercube sampling, etc.). There is also the option of halting the optimization for poor trajectories.

Using multiple starting points is a relatively simple approach that can become prohibitively costly if the objective function or constraints is expensive. In problems where the cost of fitting and evaluating a surrogate is much less than the cost of evaluating the true objective function or constraints, using multiple starting points in conjunction with surrogates is a viable approach to find multiple optima. Thus, we can investigate the use of multiple starting points to find multiple optima by solving Eq.(2). In an iterative optimization scheme, the single best solution of the starts or the multiple points resulting from multiple starting points can added per iteration. Some random sampling is needed to prevent premature convergence. This research investigates both surrogate and multi-start approaches.

4.2 Efficient Global Optimization

The Efficient Global Optimization (EGO) algorithm [7] is a sequential sampling global optimization method. It starts by fitting a surrogate that comes with a prediction uncertainty. After fitting the surrogate, the algorithm iteratively adds points to the data set in an effort to improve upon the present best sample. In each cycle, the next point to be sampled is the one that maximizes the expected improvement, $E[I(x)]$. Though another variant of the EGO algorithm uses maximizes the probability of improvement on a targeted solution (EGO-AT [12]), this work focuses on $E[I(x)]$. $E[I(x)]$ is a measure of how much improvement upon the present best sample we expect to achieve if we add a point. Rather than only searching for the optimum predicted by the surrogate, EGO will also favor points where surrogate predictions have high uncertainty. Therefore, EGO is able to balance exploitation of areas with small objective function values and exploration of areas with high uncertainty. For further details on the EGO algorithm, the reader may seek out one of the many papers on EGO, notably [7, 9].

Since EGO tries to improve upon the present best solution, it may not be possible for it to locate multiple optima if the values of the optima are very different or when some optima are very poor. For example, if a local optimum of a function is already found, the expected improvement may not be large enough to put a point in an area that contains another optimum with a poorer objective function value. For this reason, for locating multiple optima we restrict the comparisons of the EGO method to optima that are close in objective function value. Additionally, EGO is capable of adding more than one point per iteration [13, 14], but this research only considers the use of EGO in adding a single point per iteration.

5. Partitioning the Design Space

Our approach consists of splitting the space into sub-regions and assigning agents to each of these sub-regions as presented in Fig. 1. Consequently, Algorithm 1 can be thought of as the procedure followed by a single agent to find one point, that will be repeating until termination. However, in the multi-agent approach we describe here, each agent is restricted to only a sub-region 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 and more regular space, which we denote as $\mathcal{P}_i$ for the ith agent. Each agent must consider only the points in its sub-region, which are available in its internal database $(\mathbb{X}^t, \mathbb{F}^t, \mathbb{G}^t)_i$. The sub-region of an agent is defined by the position of its center c. A point in the space belongs to the sub-region with the nearest center, where the distance is the Euclidean distance. This creates sub-regions that are Voronoi cells [15]. 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 sub-regions for four agents, which requires four centers. In this example, we place the centers randomly.

Assuming that sub-regions are defined, there are two options by which surrogate approximations can be used: (1) a global surrogate is fit to the entire design space and shared among agents or (2) local surrogates are fit by each agent for each sub-region. In both cases, the surrogate chosen is the one that maximizes the accuracy in its sub-region. When a local surrogate is fit in a sub-region, there is the potential for ill-conditioning if more points are needed than are available to an agent, so 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 leaving

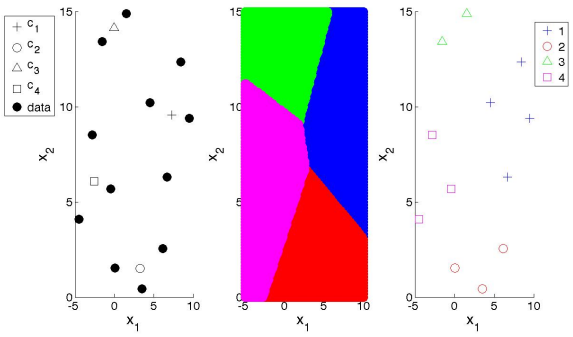

Figure 1: Two-dimensional partitioning example showing data points and four centers (left), corresponding to four agents and sub-regions (center), and assignment of data points to each agent and sub-region (right)

out a point, refitting the surrogate, and measuring the error at that point. The operation is repeated for p points in the agent's sub-region (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

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

For a global surrogate, the $PRESS_{RMS}$ is taken over the entire design space.

5.1 Finding a New Point

Once the agents have chosen surrogates or a global surrogate has been chosen for all agents, the optimization is performed to solve the problem in Eq.(2) inside the sub-region. The local optimization is repeated for multiple starting points (c.f. Sec. 4.1), and the solutions are ranked by objective function and feasibility, and points that are too near to existing data points are removed. After the ranking and removal of points, the best solution from the ranked solutions is chosen. If there are no solutions left (e.g., when all solutions are too close to already existing points), then an exploration point it added. To explore, the agent adds a point to the database that maximizes the minimum distance from already existing points within the sub-region. 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 internal database.

5.2 Dynamic Partitioning

The previous section expounds the cooperative optimization process performed by agents in a pre-partitioned space. The goal of this method is to have each agent locate a single optimum, such that the partitioning strongly depends on the topology of the space: the partitions approximate basins of attraction so that local optimizers can efficiently be used within each partition. Therefore, as a part of the cooperative optimization process, we propose a self-organizing mechanism to dynamically partition the space which adapts to the search space. By self-organizing, we mean that agents (and therefore sub-regions) 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*).

The method of space partitioning we propose focuses on moving the sub-regions' 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 sub-region 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. Then, the center of the sub-region is moved to the "best" point in the sub-region in terms of feasibility and objective function value. This is done by comparing the center at the last iteration to the last point added by the agent.

5.2.1 Merge, Split and Create Sub-regions

Once an agent has added a new point in its database and moved its center to the best point , it will check whether to split, or to merge with other ones. These operations assign search resources non-uniformly in the design space so that agents are not redundant and can efficiently rely on local optimization. Merging agents (and their sub-regions)

Table 1: Comparison of two points to decide where to place the center of a sub-region. For example, Point 1 can be the current center and Point 2 the last point added to the sub-region.

Point 1	Point 2	Center sub-region at
infeasible	infeasible, larger maximum constraint violation	Point 1
feasible	infeasible	Point 1
feasible, larger f than Point 2	feasible, smaller f than Point 1	Point 2

prevents agents from crowding the same area, allowing one agent to capture the behavior in a region. Splitting an agent is a way to make the function landscape more unimodal within the partitions. Split and merge occurs at the end of each iteration: 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' sub-regions, and then each remaining agents examines whether to split or not.

5.2.1.1 Merge Converging Agents: Agents are merged (deleted) if the centers of the agents' sub-regions 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 one with the lowest performance center is deleted. Before deletion, the deleted agent distributes its internal database points to closest neighbors.

5.2.1.2 Split Clustered Sub-regions: It is desirable to create an agent if it is found that points are clustered in two separate areas of a single agent's sub-region, as such a situation can occur if there are two optima in a subregion.

Agents are created by using k-means clustering [16] for two clusters ($k = 2$) given the points in the sub-region, 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 may not be current data points, 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 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 sub-region. The silhouette, introduced by Rousseeuw [17], 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.

5.2.1.3 Create New Agents: The agents may reach a point where there is no improvement made by the overall system in several iterations (i.e., the centers of all agents have remained at the same points). For example, this can occur when each agent has located the best point in its sub-region, 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 sub-region. In order to improve exploration, a new agent is created in the design space when there is no improvement for S iterations (i.e., the centers of the sub-regions have not moved for S 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 data points, thus forming a new agent. The design space is then repartitioned.

6. Numerical Examples

This section compares the success and efficiency of the methods presented in this paper on locating multiple candidate designs for two numerical examples. Five methods are compared: Two multi-agent methods that use dynamic partitioning but different surrogate setups, two single agent methods that add one or three points per iteration, and EGO. The methods considered are described in Table 2. In all cases, the number of function evaluations was fixed

Table 2: Description of methods

Method	Description
Agents and Partitioning: Global (Sec. 5)	Uses multi-agents with dynamic partitioning. Agents use the same global surrogate. As many points as agents are added per iteration.
Agents and Partitioning: Local (Sec. 5)	Uses multi-agents with dynamic partitioning. Agent uses local surrogate for its sub-region. As many points as agents are added per iteration.
Single (Sec. 4.1)	Single global surrogate agent adding one point per iteration. Multiple starts are used and the best point in terms of feasibility and objective function value is chosen. Exploration occurs when a start gives a solution too near an already existing point. Exploration adds a point that maximizes the minimum distance from existing points.
Single: Multiple Points Per Iteration (MPPI) (Sec. 4.1)	Single global surrogate adding 3 points per iteration by 3 start points for local optimization. Exploration occurs when a start gives a solution too near to an already existing point. Exploration adds a point that maximizes the minimum distance from existing points.
EGO (Sec. 4.2)	Global surrogate using EGO algorithm to add one point per iteration

at 100, including those required for the initial design of experiments, for both examples. The number of points added for the Single-MPPI case was set to three because it was observed that this was the mean number of points added in the multi-agent cases for the same examples. Thus, we fixed the number of points per iteration to three to provide a fairer comparison between the two methods.

As different methods listed in Table 2 add a different number of points per iteration (i.e., the Single-MPPI method adds three points per iteration, multi-agent method adds as many points as agents, and Single and EGO cases only add one point per iteration), we examine two values when comparing efficiency: number of function evaluations and number of iterations. The advantage of adding multiple points per iteration as in the Single-MPPI and multi-agent methods comes from parallelization between the processes that add the multiple points. Thus, comparing these methods to ones that add only a single point per iteration should be done based on the number of iterations rather than function evaluations. In the results presented for the two examples in this section, we present both values. Note that for the Single Agent and EGO cases the iterations and function evaluations are equal as both only add a single point per cycle.

6.1 Experimental Setup

For the example considered in this study, there are no nonlinear constraints, so only the objective function is approximated by surrogates.. The three possible surrogates are kriging surrogates with quadratic, linear, or constant trend function. 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 is 1.5 x the number of coefficients of a quadratic response surface. If the minimum number of points are not available when fitting local surrogates to the sub-regions, points are borrowed from neighboring sub-regions in the order of increasing distance to the agent center, and, if the requirement is still not met, then all available points are used.

For the agent cases with partitioning, the parameters are provided in Table 3. For distances, we consider the distance in the normalized space as a fraction of the maximum possible distance between two points in the design space (e.g., for two-dimensional problems, we normalized by $\sqrt{2}$). For example, we use this distance when considering the minimum distance between centers and the minimum distance between data points as given in Table 3.

The initial size of the DOE for both example problems was 12, with the points sampled by Latin Hypercube Sampling. In each case given in Table 2, 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 [18]. DOEs are obtained using Latin Hypercube sampling and the *maximin* criterion for five iterations.

Table 3: Multi-agent parameters for example problems

Parameter	Value
Max # of function evaluations	100
Max # of agents	6
Initial/Min # of agents	1
Min distance between agent centers	10% of max possible distance in space
Min distance between points	0.2% of max possible distance in space
Min average silhouette	0.25
Min # of points in each agent after creation	4
Stagnation threshold	3
Number of starting points for local optimization	10

6.2 Branin-Hoo Test Function

The first example is the minimization of the Branin-Hoo test function, a common benchmark test function used in surrogate-based global optimization. It is given in Eq.(5).

$$f(x) = \left(x_2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6\right) + 10\left(1 - \frac{1}{8\pi}\right)cos(x_1) + 8 \leq 0 \tag{5}$$

The domain of the function is $-5 \leq x_1 \leq 10$ and $0 \leq x_1 \leq 15$. There are three global optima of the Branin-Hoo function, for which $f = 0.40$. A contour plot of the Branin-Hoo function is shown in Fig. 2.

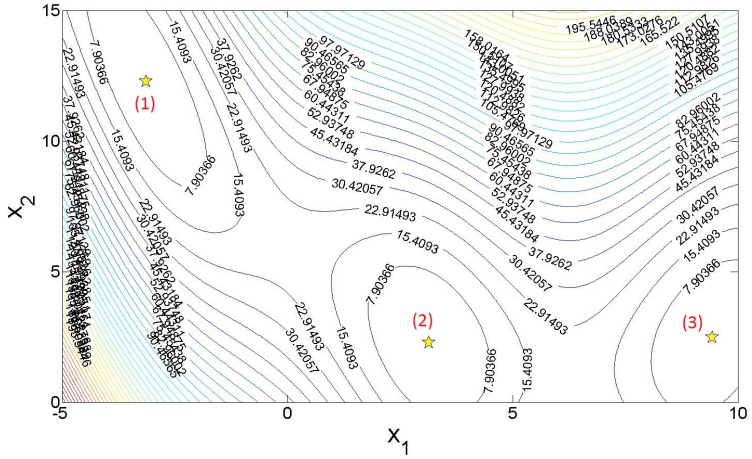

Figure 2: Contour plot of Branin-Hoo function showing three optima. For all optima, $f = 0.40$.

6.2.1 Results

For 50 repetitions, the percentage of repetitions that successfully located a solution a 1% distance from the optimum is shown in Fig. 3. This distance is the Euclidean distance normalized by the maximum possible distance between points in the design space (here, $\sqrt{2}$).

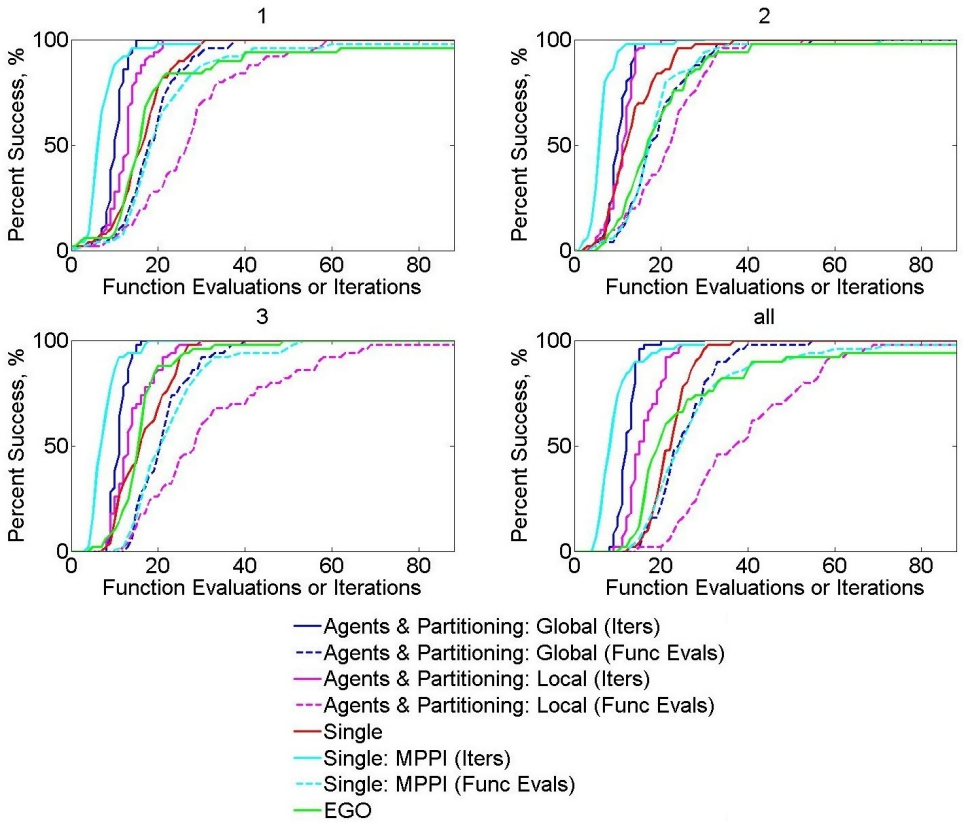

Figure 3: For the Branin-Hoo example, the percentage of 50 repetitions that found a solution within 1% distance from each optimum. For the cases that multiple points per iteration (agents and Single-MPPI) the value in terms of iterations is given by the solid lines and dashed lines for function evaluations.

The number of function evaluations shown does not include the evaluations required for the inital DOE of 12 points.

First, if we make the comparison in terms of iterations, we observed that the Single-MPPI method is quickly successful in 80% of the repetitions. However, to reach percentages greater than 90%, the multi-agent method with the global surrogate located all optima with the fewest iterations. In terms of the number of iterations, the general trend was that the multi-agent method with a global surrogate reached 100% success with the fewest iterations followed by either the multi-agent method with local surrogates or Single-MPPI, then the single agent adding one point per iteration, and finally EGO. However, it should be noted that EGO actually found solutions near to the optima with a small number of function evaluations, but required more function evaluations to put a point within 1% of all optima as it was driven to search other regions with higher expected improvement due to larger uncertainty in the surrogate.

When considering the number of function evaluations, which does not account for the parallelization in the addition of multiple points per iteration, the single agent is the most efficient while the multi-agent method with local surrogates is the least efficient with the other methods falling in between.

Figure 4 displays the placement of points in the design space after 100 total function evaluations for each method for a single repetition. It is observed that the single agent method put many points around the optima and only few points in the rest of the design space in exploration. In contrast, the Single-MPPI method both clustered points around the optima and filled the design space. The EGO method did not put nearly as many points around the optima and put many points in exploration in the design space, which is consistent with its goal of searching in areas with promising improvement.

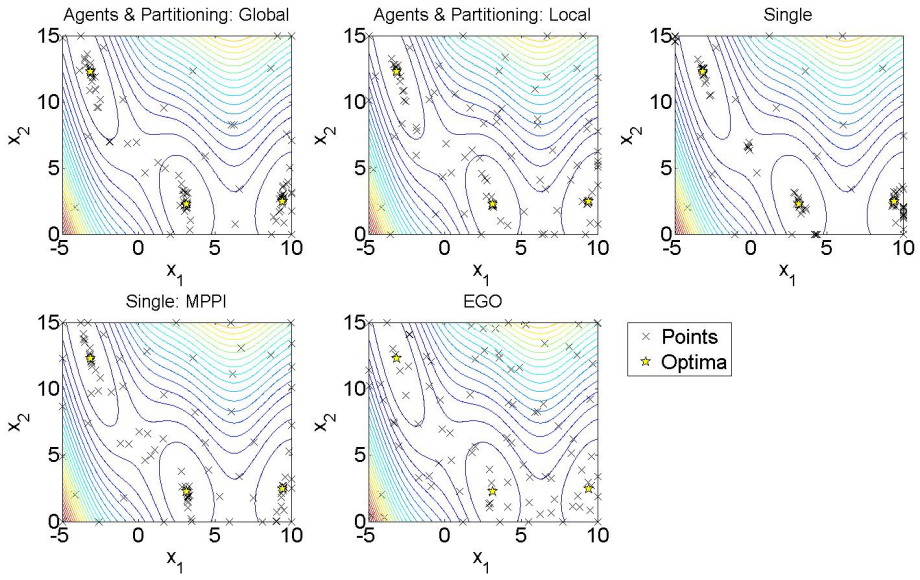

Figure 4: For the Branin-Hoo example, the plot of the points found by different methods for one repetition

The placement of points for the multi-agent cases was quite different when using global and local surrogates. While both put many points around each optimum, local surrogates resulted in many points away from the optima. This was partially due to the error in the surrogate, which predicted good objective function values away from the optima. To measure the error, we calculated the error at 1000 test points by e_{rms}. The e_{rms} normalized by the estimated range of the Branin-Hoo function is provided in Fig. 5.

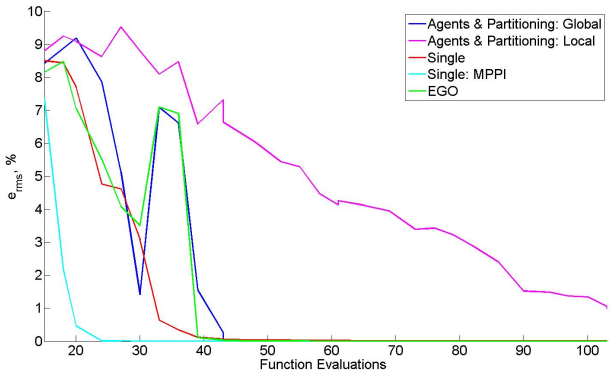

Figure 5: For the Branin-Hoo example, the error at 1000 test points with e_{rms} as the percentage of the estimated range of the function

It was observed that local surrogates were the least accurate, while the single global surrogate in the Single-MPPI method had less than 1% error after 20 function evaluations.

6.3 Sasena Test Function

The second example problem, is the minimization of the Sasena function, which was used by Sasena under the name "mystery" function due to its unknown origin[19].

$$f(x) = 2 + 0.01(x_2 - x_1^2)^2 + (1 - x_1)^2 + 2(2 - x_2)^2 + 7sin(0.5x_1)sin(0.7x_1x_2) \tag{6}$$

The domain of the function is $0 \leq x_1, x_2 \leq 5$. A contour plot of the Sasena function is shown in Fig. 6. There are four optima, as shown in the figure, but the values of optima 3 ($f = 12.7$) and 4 ($f = 33.2$) are 40% and 90% from the global optimum ($f = -1.46$) in terms of the range of the function (38.6). Optimum 2 ($f = 2.87$), which

has 11% difference from the global optimum is the only competitive optimum. Therefore, it is expected that the methods compared here are only effective at locating optimum 1 and 2.

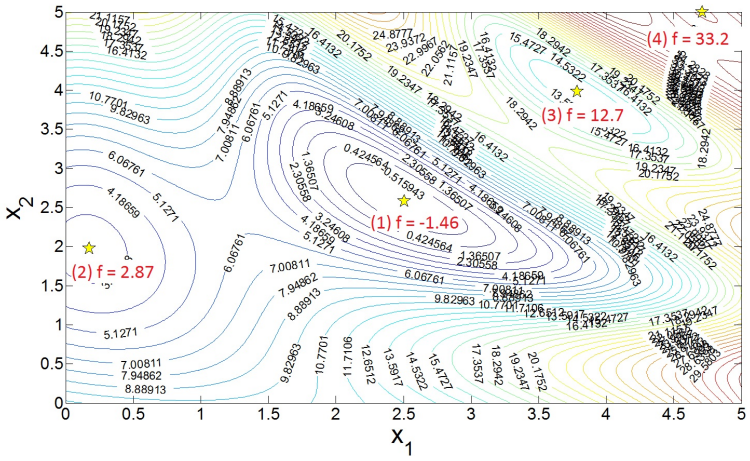

Figure 6: Contour plot of Sasena function showing four optima

In addition to its poor objective function value, optimum 4 has a small basin of attraction. In cases where the model that is being optimized has error, such small regions are may be wiped out by errors so a design in this region could be vulnerable.

6.3.1 Results

For 50 repetitions, the percentage of repetitions that successfully located a solution a 1% distance from the optimum is shown in Fig. 7. We observed very little success at locating all optima by all methods, which was not unexpected as optima 3 and 4 are poor in comparison to the top two optima.

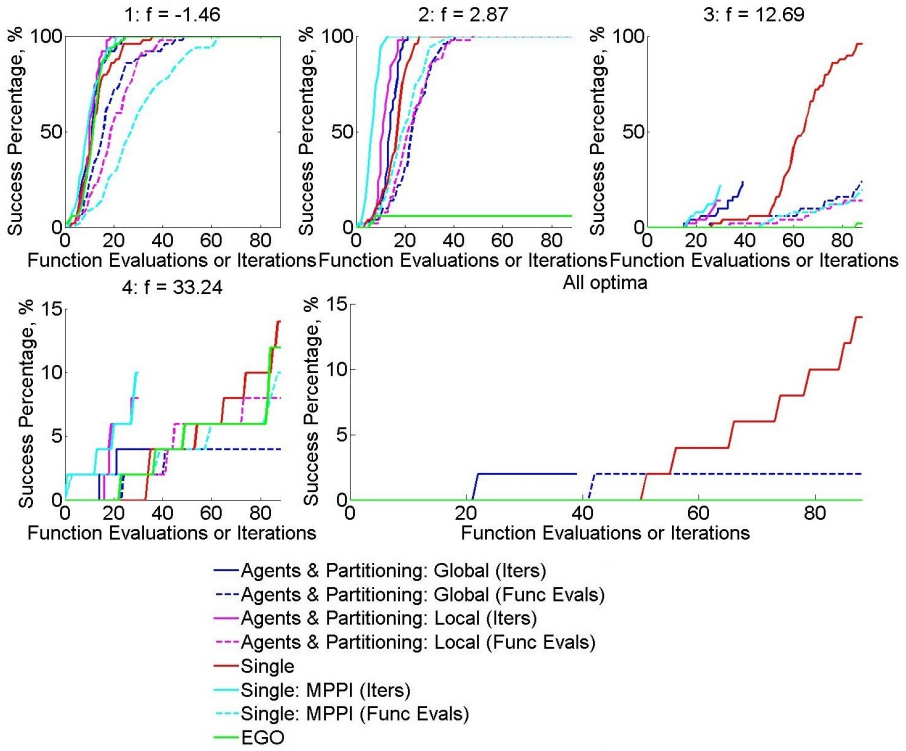

Figure 7: For the Sasena example, the percentage of 50 repetitions that found a solution within 1% distance from each optimum

When the comparison is made in terms of number of iterations, the methods that call for parallelization of the optimization processes (i.e, the multi-agent methods and Single-MPPI) outperform the one-point-per-iteration single agent and EGO algorithm. The Single-MPPI method again achieves a high percentage of successes with only a few iterations for optima 1 and 2, but the rate of success for both multi-agent cases is comparable for the global optimum. Another thing to note is the comparable performance of the multi-agent method with local surrogates compared to the global surrogate, which was not observed in the Branin-Hoo example.

It was also observed that EGO was efficient in locating the global optimum, but only had under 10% success in locating optimum 2. Based on this example, the current implementation of EGO has less potential to be successful in locating multiple optima when the optima are not almost equal.

When comparing the efficiency in terms of number of function evaluations, the one-point-per-iteration single agent is the most efficient. In fact, the single agent is able to locate optimum 3 in 90% of the repetitions. This is because the single agent locates optima 1 and 2 in early iterations, and is able to put points in the other parts of the space to locate optimum 3. The error of the surrogate approximations at 1000 test points is shown in Fig. 8. As in the Branin-Hoo example, it is observed that the single surrogate with multiple points per iterations was the most accurate and the local surrogates from the multi-agent system were the least accurate.

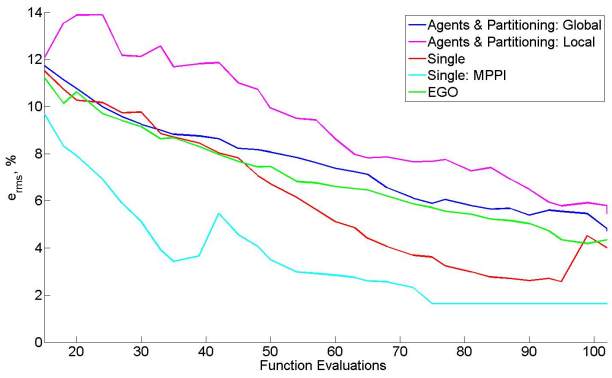

Figure 8: For the Sasena example, the error at 1000 test points with e_{rms} as the percentage of the estimated range of the function

7. Concluding Remarks and Future Work

This paper provided a comparison of the success and efficiency in locating multiple optima by different surrogate-based optimization methods. Our previously developed multi-agent method that dynamically partitions the design space was compared against the EGO algorithm and a simple method that uses multiple starts in the design space to either add one or several points in an iteration. It was observed that EGO has the potential to locate multiple optima when optima functions values are similar, while the other methods presented here have this ability for optima that are within 11% of the range of the function. In practice, this is an ideal scenario as one would not want to waste resources on searching for poor optima.

The most efficient methods of those studied here aimed to take advantage of parallel computing for optimization. The use of multiple starting points for local optimization and adding multiple points per cycle proved to be a simple yet efficient method that warrants further research. The multi-agent approach, which involves optimization in several dynamically changing sub-regions in parallel, was shown to be efficient in locating competitive optima. We observed that the error in local surrogate approximations by the multiple agents was larger compared to a global surrogate. Additionally, we did not observe that local surrogates outperformed the global surrogate in either test problem. For this reason, it may be possible to only use a global surrogate, which removes the complication of exchanging points between agents to fit local surrogates.

In the future, a more in-depth look at the advantages of using multiple points per iteration can be studied. There is the possibility to explore asynchronous agents that partition the starting points for local optimization in the space and update the global surrogate as new points are added.

8. Acknowledgments

This work has benefited from funding from Agence Nationale de la Recherche (French National Research Agency) with ANR-09-COSI-005 reference. This work was also supported by NASA under award No.NNX08AB40A

and the Air Force Office of Scientific Research under award FA9550-11-1-0066. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the ANR, NASA, or the AFOSR.

References

[1] D. Beasley, D. R. Bull, and R. R. Martin, "A sequential niche technique for multimodal function optimization," *Evolutionary computation*, vol. 1, no. 2, pp. 101–125, 1993.

[2] C. Hocaoglu and A. C. Sanderson, "Multimodal function optimization using minimal representation size clustering and its application to planning multipaths," *Evolutionary Computation*, vol. 5, no. 1, pp. 81–104, 1997.

[3] S. Nagendra, D. Jestin, Z. Gurdal, R. T. Haftka, and L. T. Watson, "Improved genetic algorithm for the design of stiffened composite panels," *Computers & Structures*, vol. 58, no. 3, pp. 543–555, 1996.

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

[5] A. Torn and A. Zilinskas, "Global optimization," in *Lecture Notes in Computer Science 350*, Springer Verlag, 1989.

[6] A. Torn and S. Viitanen, "Topographical global optimization using pre-sampled points," *Journal of Global Optimization*, vol. 5, pp. 267–276, Oct. 1994.

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

[8] N. M. Alexandrov, J. E. Dennis Jr, R. M. Lewis, and V. Torczon, "A trust-region framework for managing the use of approximation models in optimization," *Structural Optimization*, vol. 15, no. 1, pp. 16–23, 1998.

[9] M. Schonlau, *Computer experiments and global optimization.* University of Waterloo, 1998.

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

[11] J. Kleijen, *Design and analysis of simulation experiments*. Springer, 2008.

[12] A. Chaudhuri, R. Haftka, and F. Viana, "Efficient Global Optimization with Adaptive Target for Probability of Targeted Improvement," in *8th AIAA Multidisciplinary Design Optimization Specialist Conference*, (Honolulu, HI), pp. 1–13, American Institute of Aeronautics and Astronautics, Apr. 2012.

[13] F. A. C. Viana, R. T. Haftka, and L. T. Watson, "Why not run the efficient global optimization algorithm with multiple surrogates?," in *51th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, (Orlando, FL,USA), 2010.

[14] J. Janusevskis, R. Le Riche, D. Ginsbourger, and R. Girdziusas, "Expected improvements for the asynchronous parallel global optimization of expensive functions: Potentials and challenges," in *Learning and Intelligent Optimization*, pp. 413–418, Springer, 2012.

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

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

[17] P. J. Rousseeuw, "Silhouettes: a graphical aid to the interpretation and validation of cluster analysis," *Computational and Applied Mathematics*, vol. 20, pp. 53–65, 1987.

[18] MATLAB, *version 7.9.0.529 (R2009b)*, ch. fmincon. Natick, Massachusetts: The MathWorks Inc., 2009.

[19] M. J. Sasena, *Flexibility and Efficiency Enhancements for Constrained Global Design Optimization with Kriging Approximations by*. PhD thesis, University of Michigan, 2002.