George H. Cheng
Product Design and Optimization Lab (PDOL),
School of Mechatronic Systems Engineering,
Simon Fraser University,
250-13450 102 Avenue,
Surrey, BC V3T0A3, Canada

Adel Younis
Product Design and Optimization Lab (PDOL),
School of Mechatronic Systems Engineering,
Simon Fraser University,
250-13450 102 Avenue,
Surrey, BC V3T0A3, Canada

Kambiz Haji Hajikolaei
Product Design and Optimization Lab (PDOL),
School of Mechatronic Systems Engineering,
Simon Fraser University,
250-13450 102 Avenue,
Surrey, BC V3T0A3, Canada

G. Gary Wang[1]
Product Design and Optimization Lab (PDOL),
School of Mechatronic Systems Engineering,
Simon Fraser University,
250-13450 102 Avenue,
Surrey, BC V3T0A3, Canada
e-mail: gary_wang@sfu.ca

Trust Region Based Mode Pursuing Sampling Method for Global Optimization of High Dimensional Design Problems

Mode pursuing sampling (MPS) was developed as a global optimization algorithm for design optimization problems involving expensive black box functions. MPS has been found to be effective and efficient for design problems of low dimensionality, i.e., the number of design variables is less than 10. This work integrates the concept of trust regions into the MPS framework to create a new algorithm, trust region based mode pursuing sampling (TRMPS2), with the aim of dramatically improving performance and efficiency for high dimensional problems. TRMPS2 is benchmarked against genetic algorithm (GA), dividing rectangles (DIRECT), efficient global optimization (EGO), and MPS using a suite of standard test problems and an engineering design problem. The results show that TRMPS2 performs better on average than GA, DIRECT, EGO, and MPS for high dimensional, expensive, and black box (HEB) problems. [DOI: 10.1115/1.4029219]

1 Introduction

In engineering, computational tools such as finite element analysis (FEA) and computational fluid dynamics (CFD) are widely used as part of the product design process. However, FEA and CFD models are complex and often computationally expensive. Design engineers also desire to identify the global optimum since practical engineering problems often involve multiple local minima. Those FEA and CFD models are commonly viewed as black box functions, and are not directly solvable by classical gradient-based optimization methods. Gradient information can be obtained via finite differencing, but such information can be costly to compute, requiring linearly increasing numbers of model evaluations as the dimensionality of the problem increases as well as frequent gradient recomputation. Also, Ref. [1] states that the gradients computed from these models can be unreliable or noisy and thus may not allow for accurate optimization, although this is model-dependent and only occurs in certain cases. In addition, despite the ever increasing computing power, the computational challenges of design optimization on expensive simulations remain, if not intensified. This is mainly due to the fact that more sophisticated and complex analysis and simulation models evolve to take advantage of the increased computing power.

Taking the vehicle crash simulation and optimization for an example, in 2001 Gu stated that "it takes about 36 h for single simulation in SGi Origin 2000" [2]. Although computational power has increased tremendously since Gu's publication, the complexity of computational models has also increased commensurably and as a result the simulation time has not been significantly reduced. Duddeck in a rather recent publication [3] stated that a typical automotive crash simulation on 8 central processing units (CPUs) takes 20 h. In short, with increasing computing power comes the opportunity to build more accurate, complex, and expensive models which require algorithms that can efficiently handle black box, computationally expensive problems with a global optimum. The area of global optimization has been a hotbed of activity in the recent past. Some popular gradient-free global optimization algorithms are the GA [4], simulated annealing (SA) [5], particle swarm optimization (PSO) [6], EGO [7], and dividing rectangles search method (DIRECT) [8]. GA and PSO are population-based evolutionary algorithms that use a combination of random sampling and various point elimination and selection strategies to iteratively improve the fitness of the population. GA is inspired by genetic evolution, and employs the genetic concepts of reproduction, mutation, crossover, and fitness selection as part of its evolutionary strategy. PSO, on the other hand, is inspired by the behavior of bird swarms and social groups. Instead of the genetic analogue employed by GA, PSO uses the analogy of agent movement and behavior; and all individuals determine their actions partially or wholly based on the status and behavior of other individuals. Evolutionary algorithms such as GA and PSO have proven to be very versatile and popular. In contrast, SA uses a simple strategy to help avoid the search becoming stuck in a local minimum [5]. The strategy is to allow the acceptance of worse solutions according to a gradually decreasing probability over the course of the search, which makes SA probabilistic and allows SA to effectively solve global optimization problems. DIRECT, in direct contrast to evolutionary algorithms, is a deterministic global optimization algorithm that relies primarily on effective structured sampling to advance the search. EGO, in contrast to the other algorithms mentioned, is a metamodel-based optimization method that relies on the use of a Kriging metamodel. The Kriging method is used to construct an approximation model of the search space using points evaluated with the black box. The approximation model is then used to identify candidate points in areas of large modeling error and areas of promise of finding a better solution.

Despite an abundance of global optimization algorithms, there is still a dearth of algorithms that are capable of optimizing HEB

[1]Corresponding author.
Contributed by the Design Automation Committee of ASME for publication in the Journal of Mechanical Design. Manuscript received January 16, 2014; final manuscript received November 17, 2014; published online December 17, 2014. Assoc. Editor: Christopher Mattson.

problems. High dimensionality is referred when the number of variables is larger than ten in the context of design involving expensive functions [9]. The adjective "expensive" is a qualitative classification of simulations that have high computational cost. The word "black box" implies that the objective function is strictly an input–output function whose internal composition is not accessible or implicit. For the HEB class of problems, the number of function evaluations cannot be too high due to the high computational cost of the problems. The high dimensionality of the problems increases the size and complexity of the search space which makes modeling and searching of the space a challenge. Also, the black box nature of the problems means that gradient information is not readily available. Although all of the previously mentioned algorithms are capable of global optimization, few are capable of addressing HEB problems. Population-based algorithms such as PSO tend to require a large number of function evaluations [10]. SA and DIRECT suffer from slower convergence in high dimensional problems, and EGO has high algorithmic computational cost for high-dimensional problems (demonstrated in this work).

In dealing with optimization with expensive simulations, our group developed the MPS method for solving global optimization problems with expensive black-box functions [11,12]. MPS is a metamodel-based global optimization algorithm, in the same class of algorithms as EGO. In general, MPS outperforms other popular global optimization methods such as GA in expensive black-box problems with low dimensionalities [13]. MPS, however, has drawbacks that need to be overcome in order to be fully applied in real-world industrial problems. One of the biggest problems facing MPS is its low performance on high-dimensional problems with expensive functions due to the high numbers of function evaluations required and the high likelihood of exceeding available memory [13]. In addition, the efficiency of MPS should be improved in general to make it more attractive for industrial applications. In light of these issues, the authors developed a trust region based MPS (TRMPS) algorithm to tackle high dimensional problems. Roughly speaking, a trust region has the following basic features: (1) a trust region is a subregion of a space, (2) the size of the trust region is dynamically adjusted, and (3) a trust region is normally defined around a center point. The trust region method has been well studied before [14]. In particular, Refs. [15–17] have applied the trust region methodology for global optimization. Previous work on the application of trust regions to global optimization have been focused either on the direct usage of trust regions for identifying promising solutions [15], or the use of trust regions to limit the scope of construction of the metamodels [16,17]. The work presented in this paper is a novel application of the trust region methodology that differs from these previous works. First, in TRMPS the trust regions are not used to directly identify candidate solutions. Instead, the trust regions are used to restrict the scope of the metamodels. Second, two trust regions are used instead of one and the behavior of the regions differs significantly from past work, such as Refs. [16,17]. Whereas in Refs. [16,17] the trust region is adjusted to increase the accuracy of the approximation model, this work uses trust regions to guide metamodeling toward regions with more promising solutions. This work will define two trust regions that have all of the above features and their sizes are dynamically adjusted in tandem for global optimization.

A preliminary version of the TRMPS algorithm was presented in Ref. [18], which outlined the new algorithm and compared the performance of TRMPS against MPS in a few preliminary tests. Because of poor performance of the MPS algorithm on high dimensional problems, MPS was modified to enable optimization on problems of high dimensionality, and thus enable performance comparisons with TRMPS. While the results presented in Ref. [18] are promising and demonstrate the superiority of TRMPS over MPS, TRMPS was not evaluated with respect to other popular global optimization algorithms. Our experiments indicated at the time that TRMPS performed poorly compared to GA. We have subsequently made major improvements and modifications to the preliminary TRMPS algorithm. This paper presents an improved version of the algorithm called TRMPS2, and benchmarks its performance relative to MPS, GA, DIRECT, and EGO.

For the sake of brevity, some concepts are expressed using concise terminology in this paper. A metamodel is a mathematically known "white-box" function that is used to approximate a black-box function. Points evaluated by a metamodel are referred to as cheap points; points evaluated by the original black-box function are referred to as expensive points.

2 Review of MPS

Before introducing TRMPS2, we first briefly review the basic MPS algorithm. For further details on the material in this section, please refer to Refs. [10] and [11].

2.1 MPS Algorithm. MPS relies primarily on stochastic sampling and metamodeling to perform optimization. At the beginning of the MPS algorithm a small set of sample points are randomly generated. The algorithm checks to see if all of the constraints are satisfied and points are generated until n_{kk}

$$n_{kk} = \frac{(n_v + 1)(n_v + 2)}{2} + 1 - n_p \tag{1}$$

number of feasible points are accumulated, where n_v is the number of variables, and n_p is set as n_v in this work. Then the cost function is evaluated at these feasible points to produce the set of expensive points $\{y\}$. A metamodel is generated by fitting $\{y\}$ onto a radial basis function (RBF), which takes the form of the following equation:

$$\hat{f}(x) = \sum_{i=1}^{m} \alpha_i ||x - x_i|| \tag{2}$$

where m is the number of points in $\{y\}$, α_i is the weight of the function or the objective function value of point x_i, and the set of points x_i are the expensive points used to build the model. N feasible points are then randomly generated in the search space after verifying constraints, and are then evaluated using the generated metamodel to create the set of cheap points $\{x\}$. $\{x\}$ is then sorted based on function values and then subdivided into contours, with each contour representing a slice of $\{x\}$ within a certain function value range. n_k is the number of points per contour. Expensive samples are then drawn from the contours, with the process controlled by a speed control factor r. R^2 values from the regression and the parameter Diff, which defines the maximum relative difference between the actual and approximated function values, are used to judge the quality of the local approximation. Diff is calculated as

$$\text{Diff} = \max\left\{ \left| f_m^{(i)} - f^{(i)} \right|, i = 1, \ldots, j \equiv \frac{(n_v + 1)(n_v + 2)}{2} + 1 + \frac{n_v}{2} \right\} \tag{3}$$

where f_m are the function values of the points fitted onto the quadratic model and f are the function values of the expensive samples. With a small value of ε_d, Diff $< \varepsilon_d$ functions as a quality criterion. A quadratic approximation model is built for the region around the current optimum point (referred to as the current function mode) and is expressed as

$$q = \beta_0 + \sum_{i=1}^{n} \beta_i x_i + \sum_{i=1}^{n} \beta_{ii} x_i^2 + \sum_{i<j} \sum_{j=1}^{n} \beta_{ij} x_i x_j \tag{4}$$

where $\boldsymbol{\beta}$ are model coefficients. The entire process iterates until certain stopping criteria are met. For a more detailed treatment please refer to Refs. [11,13].

2.2 Performance Issues of MPS for High Dimensional Problems. The effectiveness of MPS on optimizing problems of lower dimensions has been demonstrated in Ref. [13]. However, the performance of MPS decreases dramatically with nonlinear high dimensional problems. Because MPS generates a RBF metamodel in each iteration using all of the expensive points available and because the number of expensive points increases with each iteration of MPS, the computational cost of MPS rises with each iteration. Not only does MPS require more CPU time with every passing iteration but it also requires more memory to store the large matrices. In addressing the memory overflow issue in this work, during each iteration of MPS the set of all expensive points is sorted randomly and a smaller subset of expensive points with ten times the number of variables (i.e., $10*n_v$) is selected for metamodeling. This allows MPS to run simulations on high dimensional problems without memory overflow.

3 Trust Region Based MPS

To improve the performance of TRMPS for solving high dimensional problems, this work develops an improved version, called TRMPS2.

3.1 Definition of Trust Region: Dual Hyper Cubes. We define two hyper cubes S and B. The region enclosed in hyper cube S is referred to as trust region S (TR_S), and the region between the two hyper cubes S and B is referred to as TR_B. Thus, TR_B occupies the area of B minus the area of S, i.e., $TR_B = B - S$. At the beginning of the optimization, MPS processes start from both trust regions. If the search improves through the discovery of a better optimum point, then S expands to avoid being trapped into a local minimum region and B contracts to exploit the promising region. If the search does not improve in a certain number of iterations, S contracts and B expands. The strategy is to use this dynamic balance between exploitation and exploration to more efficiently sample the search space. The behavior of the two trust regions is similar to the double sphere strategy described in Ref. [12].

3.2 Guided Sampling. At the beginning of MPS there is an initial sampling procedure that uniformly samples the search space. The number of points generated by this process is expressed by Eq. (1). In TRMPS2 this exact procedure is performed for both hyper cubes. From Eq. (1) it is clear that as n_v increases, the number of points required for initial sampling also increases. For a problem of 20 variables the number of points required for the initial sampling procedure is 212. Clearly this process is costly for problems of high dimension, but is necessary in MPS since Eq. (1) represents the minimum number of expensive points required to build the local quadratic approximation model expressed by Eq. (4). In TRMPS2 the initial sampling operation has been replaced by a guided sampling procedure that dramatically reduces the initial sampling requirements and takes advantage of the adaptive properties of the dual trust region method. A small number of points $n_{ip} = 10$ are uniformly and randomly sampled in each hyper cube initially. Thereafter, sampling is guided by the dual trust region strategy, which is to be described in more detail in the TRMPS2 Algorithm section.

3.3 Low Function Value (LFV) Selection. The modification applied to MPS to resolve the memory issue, described in the MPS section above, is also applied to TRMPS2. Moreover, instead of randomly selecting the $10*n_v$ points for building the RBF metamodel, the points are chosen on the basis of LFV in TRMPS2. The contour-based discriminative sampling strategy in Ref. [19] by Fu and Wang is applied to balance the selection of modeling points between regions of lower and higher function values. Fu and Wang's strategy involves the three steps of discretization, contourization, and sampling. In this application, only the contourization and sampling steps are applied. The discretization step involves the generation of uniformly sampled points in a given space and as we already have the points that we wish to select from, this step is not needed. Thus, the LFV method for a given trust region TR. Defined by bounds $[x_{lb}, x_{ub}]$ begins with a set of nonuniformly distributed expensive samples $S \in [x_{lb}, x_{ub}]$. The first step is contourization, which partitions S into k contours, where each contour except for the kth contour has $nk = \text{floor}(N_S/k)$ points, where N_S is the number of points in S. The partitioning occurs according to the function values of the points in S. The first nk points with the highest function values are allocated to the first contour E_1, the next nk points to E_2, and the process continues until the last contour E_k. In the second step, samples are drawn from the contours in two stages. In the first stage, the numbers of samples to be drawn from the contours $\{m_1, \ldots, m_k\}$ are determined via probabilities $\{P_1(E_1), \ldots, P_k(E_k)\}$, where $\sum_{i=1}^{k} m_i = 10^* n_v$ and probability P_i is proportional to the average function value of the points in E_i with $i = 1, \ldots, k$. Thus, $P_i = 1 - \left[\sum_{j=1}^{i} f_{\text{avg}}(E_j) / \sum_{j=1}^{k} f_{\text{avg}}(E_j)\right]$, where $f_{\text{avg}}(E_j)$ is the average function value of the points in E_j. In the second stage, m_i samples are randomly drawn from E_i, where $i = 1, \ldots, k$.

3.4 Trust Region Based Local Optimization. MPS currently relies on the fmincon(.) function from the Optimization Toolbox in MATLAB™ to perform local optimization. Although fmincon(.) performs well, it does not take advantage of the properties of the quadratic approximation model that is constructed during the local optimization phase of MPS. The quadratic model allows the construction of a Hessian matrix that, when positive definite, can enable discovery of the local optimum in one step. For TRMPS2, a new trust region based local optimizer is implemented that takes advantage of this property while also guarding against the case of the negative definite Hessian. We note that this trust region algorithm is only used for this local optimization step, and is separate from the trust region strategy employed for TRMPS2 global optimization.

3.4.1 Hessian Construction. Equation (4) is the equation of the quadratic model used for local optimization. Taking the derivative of Eq. (4) results in the components of the gradient of the fitting function

$$\frac{\partial f}{\partial x_i} = \beta_i + 2\beta_{ii} x_i + \sum_{j=1}^{n} \beta_{ij} x_j \tag{5}$$

Taking the derivative of Eq. (5) results in the components of the Hessian of the quadratic model

$$\frac{\partial^2 f}{\partial x_i \partial x_j} = \beta_{ij} \tag{6}$$

$$\frac{\partial^2 f}{\partial x_i^2} = 2\beta_{ii} \tag{7}$$

Assembling Eqs. (6) and (7) into a matrix results in the Hessian of the quadratic model

$$H = \begin{bmatrix} 2\beta_{11} & \cdots & \beta_{1j} \\ \vdots & \ddots & \vdots \\ \beta_{i1} & \cdots & 2\beta_{ii} \end{bmatrix} \tag{8}$$

Thus, the Hessian matrix can be directly assembled from the coefficients of the quadratic model.

There are three cases that need to be addressed when performing local optimization on the fitting function.

3.4.2 Case 1: Positive Definite Hessian. When the Hessian matrix is positive definite the quadratic model is convex. Thus, the

Newton method can be used to obtain the local minimum with only one iteration since the Hessian does not need to be updated. This is very efficient, and case 1 is the preferred mode. The basic algorithm used to find the local minimum for this case is as follows.

(1) Build Hessian and gradient vector from the coefficients of the fitting function.
(2) Calculate search direction ($\boldsymbol{d}^{(0)} = -\boldsymbol{H}^{-1}\boldsymbol{c}^{(0)}$).
(3) Calculate step size using a 1D search method.
(4) Update design ($\boldsymbol{x}^{(1)} = \boldsymbol{x}^{(0)} + \alpha_0\boldsymbol{d}^{(0)}$).

where $\boldsymbol{c}$ is the gradient vector, $\boldsymbol{d}$ is the search direction, and α is the step size.

3.4.3 Case 2: Negative Definite Hessian. When the Hessian matrix is negative definite the quadratic model is concave. In this case, the quasi-Newton step of Sec. 3.4.2 cannot be used. Instead, we implement a trust region based local optimizer similar to the one presented in Ref. [14]. The algorithm steps are as follows:

Step 0: Initialization

$$\begin{aligned} &\boldsymbol{x}_1 \in \mathbb{R}^n, \quad \Delta_1 > 0, \quad \epsilon > 0, \quad \boldsymbol{B}_1 \in \mathbb{R}^{nxn}_{\text{symmetric}} \\ &0 < \tau_3 < \tau_4 < 1 \\ &0 \le \tau_0 \le \tau_2 < 1, \quad \tau_2 > 0, \quad k = 1 \end{aligned} \tag{9}$$

where $\boldsymbol{x}_1$ is the initial starting point, Δ_1 is the initial trust region size, ϵ is the convergence criterion magnitude, $\boldsymbol{B}_1$ is the initial Hessian matrix, and τ_0–τ_4 are constants. τ_0 controls the acceptance or rejection of the new point for the next iteration, τ_1 controls the maximum extent of trust region expansion in a given iteration, τ_2 influences whether to shrink or expand the trust region, and τ_3 and τ_4 control the minimum and maximum extent of trust region shrinkage in a given iteration. The default choices are $\tau_0 = 0$, $\tau_1 = 2$, $\tau_2 = \tau_3 = 0.25$, $\tau_4 = 0.5$, $\epsilon = 0.001$, and $\Delta_1 = 1$. k is the iteration number.

Step 1: Initial step determination
Solve the trust region subproblem, defined as

$$\begin{aligned} &\min_{\boldsymbol{s}_k \in \mathbb{R}^n} \boldsymbol{g}_k^T\boldsymbol{s}_k + 0.5\boldsymbol{s}_k^T\boldsymbol{B}_k\boldsymbol{s}_k = m_k(\boldsymbol{s}_k) \\ &\text{s.t.} \|\boldsymbol{s}_k\|_2 \le \Delta_k \end{aligned} \tag{10}$$

where $\boldsymbol{s}_k$ is a solution to the subproblem in the kth iteration of the trust region method, and $\boldsymbol{g}_k = \nabla f(\boldsymbol{x}_k)$. At this step, $k = 1$. This subproblem can be solved using a number of methods, such as the Steihaug Toint method [14]. Then, go to step 3.

Step 2: Convergence criterion
If $\Delta_k \le \epsilon$, stop.
Else, go to step 3.

Step 3: Solve trust-region subproblem (step determination)
Solve the trust region subproblem defined in Eq. (10), with result $\boldsymbol{s}_k$. Go to step 4.

Step 4: Acceptance of the Trial Point
Calculate the approximation accuracy ratio r_k

$$r_k = \frac{f(\boldsymbol{x}_k) - f(\boldsymbol{x}_k + \boldsymbol{s}_k)}{m_k(\boldsymbol{x}_k) - m_k(\boldsymbol{x}_k + \boldsymbol{s}_k)} \tag{11}$$

where f is the objective function of the local optimization problem and m_k is the approximation model about the point $\boldsymbol{x}_k$ defined as the 2nd order Taylor series

$$\begin{aligned} m_k(\boldsymbol{x}) = &f(\boldsymbol{x}_k) + \nabla f(\boldsymbol{x}_k)^T(\boldsymbol{x} - \boldsymbol{x}_k) \\ &+ 0.5(\boldsymbol{x} - \boldsymbol{x}_k)^T D^2 f(\boldsymbol{x}_k)(\boldsymbol{x} - \boldsymbol{x}_k) \end{aligned} \tag{12}$$

If $r_k \le \tau_0$, then the approximate model is not accurate enough, so define $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k$.

Else, define $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{s}_k$.
Go to step 5.

Step 5: Trust-region radius update
Choose Δ_{k+1} that satisfies

$$\Delta_{k+1} \in \begin{cases} [\tau_3\|s_k\|_2, \tau_4\Delta_k] & \text{if } r_k < \tau_2 \\ [\Delta_k, \tau_1\Delta_k] & \text{otherwise} \end{cases} \tag{13}$$

Go to step 2.

3.5 Process of TRMPS2

3.5.1 TRMPS2 Settings. $R_{\min} = 0.01$ and $R_{\max} = 1$ are the minimum and maximum size, respectively, of both hyper cubes S and B with respect to the size of the search space. Note that $R_{\min}$ and $R_{\max}$ are not related to the statistical R^2 measure. Stall iterations = 5 is the maximum number of iterations of no improvement in the search that TRMPS2 tolerates before resizing the hyper cubes, and is also the number of iterations that TRMPS2 waits after the dual regions reach steady state before switching to the MPS mode. $k_{\text{reduction}} = 0.7$ represents the reduction or expansion in size of the hyper cubes upon resizing. $R_{\text{S,initial}} = 0.25$ and $R_{\text{B,initial}} = 1$ are the initial sizes of hyper cubes S and B, respectively, relative to the size of the search space.

3.5.2 TRMPS2 Algorithm. We define sets of points $\{y\}$, $\{y_S\}$, and $\{y_B\}$, where $\{y_S\}, \{y_B\} \subseteq \{y\}$ and $\{y_S\} \cap \{y_B\} = \emptyset$. By adding elements to either $\{y_S\}$ or $\{y_B\}$ or any other sets of expensive points, we imply automatic addition of the same elements to $\{y\}$ unless $\{y\}$ already contains those elements. Furthermore, every set of points has a corresponding set of function values where actions on the set of points are mirrored on the set of function values. Thus, $\{y\}$, $\{y_S\}$, and $\{y_B\}$ have corresponding sets of function values $\{f_y\}$, $\{f_{y_S}\}$, and $\{f_{y_B}\}$.

Step 1: Initial sampling
Sample n_{ip} uniform random points $\{y\}$ in the design space, and evaluate $\{y\}$ with the black box function to obtain function values $\{f_y\}$.
For each *pt* in $\{y\}$,
If *pt* is within TR_S, place *pt* in $\{y_S\}$,
Else if *pt* is within TR_B, place *pt* in $\{y_B\}$.

Step 2: Metamodel construction
Construct the RBF model according to Eq. (2) using expensive points $\{y_S\}$ in TR_S.

Step 3: Metamodel point fitting
Sample $n_{0s} = 10,000$ uniform random points in TR_S and evaluate them with the RBF model, with resultant point set $\{x_S\}$.

Step 4: Contour selection
Use the discriminative sampling procedure in MPS, as described in Ref. [11], to select a subset $\{x_{Si}\}$ of $\{x_S\}$. Evaluate $\{x_{Si}\}$ with the black box to obtain corresponding set $\{f_{x_{Si}}\}$. Place $\{x_{Si}\}$ in $\{y_S\}$.

Step 5: Quadratic model construction
If the number of points in $\{y_S\} > n_{kk}$, where n_{kk} is from Eq. (1), build quadratic model with Eq. (4) using the n_{kk} points in $\{y_S\}$ closest to the current optimum. This set of points is referred as $\{y_{kk}\}$. Otherwise, skip step 6 and go to step 7.

Step 6: Local optimization
If $R^2 \approx 1$, sample $nv/2$ expensive points in a region defined by $\{y_{kk}\}$ and add to the set. Reconstruct quadratic model using $\{y_{kk}\}$ and recalculate R^2. Calculate Diff using Eq. (3).
If $R^2 \approx 1$ and Diff $< \varepsilon_d$, perform local search using the trust region based local optimizer with output x_l. Evaluate x_l with the black box.
If x_l is within the region defined by $\{y_{kk}\}$, *terminate*.
Else, add x_l to $\{y\}$.

Step 7: B region
Perform steps 2–6 for TR_B.

Step 8: Region updates
If search improved this iteration, $R_B = R_B \cdot k_{reduction}$, $R_S = R_S / k_{reduction}$.
Else if no improvement for stalliterations, $R_S = R_S \cdot k_{reduction}$, $R_B = R_B / k_{reduction}$.
$\{y_S\},\{y_B\} = \emptyset$
For each *pt* in $\{y\}$,
If *pt* is within TR_S, place *pt* in $\{y_S\}$,
Else if *pt* is within TR_B, place *pt* in $\{y_B\}$.
Go to step 2.

If the maximum number of function evaluations reached, then exit algorithm.

4 MATLAB Simulations and Results

4.1 Test Methodology. Two separate sets of tests were performed to benchmark the performance of TRMPS2. The first set of tests benchmarks TRMPS2 against GA, MPS, and DIRECT with a limit of 5000 function evaluations. For the first set, fifteen high dimensional unconstrained simulation problems derived from seven different mathematical functions and one high dimensional engineering application problem are selected, and the problems range from 10 variables to 30 variables. For TRMPS2, GA, and MPS, 30 runs of each problem are carried out to reduce the effect of random variation on the simulation results. For DIRECT, only one run is performed for each problem as DIRECT is a deterministic algorithm, and produces the same result given the same parameters. The second set of tests benchmark TRMPS2 against EGO on three problems. Ten runs of each problem are performed with a maximum limit of 1000 function evaluations. In addition, all runs of the second set of tests are performed on the same computer.

EGO is not included in the first set of tests because it is found to be too computationally expensive for problems of high dimension and could not perform the simulations in a reasonable amount of time. In order to demonstrate the high computational cost of EGO and thereby justify its inclusion in a separate set of tests, the run time of EGO and TRMPS2 are recorded on the same machine for a smaller set of test problems.

It is to be noted that no commercial algorithms are benchmarked in this paper. If available, commercial stochastic algorithms such as the Pointer method in Isight [20] and the SHERPA algorithm in HEEDS [21] are good candidates for comparison.

4.2 GA Benchmark Algorithm. The implementation of GA used for the benchmarks is the version found in the Global Optimization Toolbox of MATLAB™ by Houck et al. [22]. Note that there are a large number of different implementations and variations of GA of differing performance, thus this testing only seeks to make comparisons with one particular implementation of GA in order to illustrate the strengths and weaknesses of TRMPS2. Also, GA is an exploration-only algorithm whereas all other algorithms tested, including TRMPS2, are exploration– exploitation methods.

The settings used for GA are partly obtained from Ref. [13], with crossover fraction = 0.6, mutation fraction = 0.05, population size = 100, and stall generation limit = 49. The settings have been tuned for optimal performance of the GA algorithm and deviation from these settings is likely to result in decreased performance. The stall generation limit has been set to 49 in order to limit the number of function evaluations (nfe) to a maximum of 5000, in accordance with the limits placed on MPS and TRMPS2.

4.3 EGO Benchmark Algorithm. The EGO [1] algorithm is based on the idea of fitting a Kriging metamodel to data collected by evaluating the objective function at a few points. Then, EGO balances between finding the minimum of the metamodel and improving the approximation by sampling at locations where the prediction error may be high. The code for EGO is from the Surrogates Toolbox V3.0 by Felipe Viana [23], and the default settings for EGO are used in this work.

4.4 Direct Benchmark Algorithm. The DIRECT [8,24] algorithm is a sampling optimization algorithm introduced by Jones in 1993 by modifying the Lipschitzian optimization to solve difficult global optimization problems. The DIRECT code used is the Version 4.0 MATLAB code obtained from Ref. [25]. The default parameters for DIRECT are used in our tests, as the parameters mostly affect convergence rather than the efficacy of the algorithm.

4.5 High Dimensional Test Problems

4.5.1 16 Variable Function (F16)

$$f_{F16}(x) = \sum_{i=1}^{16}\sum_{j=1}^{16} a_{ij}\left(x_i^2 + x_i + 1\right)\left(x_j^2 + x_j + 1\right); \quad i,j = 1,2,\cdots,16; \quad x \in [-1,1] \tag{14}$$

$$a_{ij} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{15}$$

Global optimum value [11]: 25.875

4.5.2 10D, 20D, and 30D Rosenbrock Functions (R10, R20, and R30, respectively)

$$f(x) = \sum_{i=1}^{n-1}\left(100\left(x_{i+1} - x_i^2\right)^2 + \left(x_i - 1\right)^2\right); \quad x \in [-5,5] \tag{16}$$

Global optimum value [13]: 0

4.5.3 10D, 20D, and 30D SUR-T1-14

$$f(\boldsymbol{x}) = (x_1 - 1)^2 + (x_n - 1)^2 + n\sum_{i=1}^{n-1}(n-i)(x_i^2 - x_{i+1})^2; \quad -3 \le x_i \le 2; \quad i = 1,\ldots,n \tag{17}$$

Global optimum value [26]: 0

4.5.4 10D, 20D, and 30D PUR-T1-13

$$f(\boldsymbol{x}) = \left[\sum_{i=1}^{n} i^3 (x_i - 1)^2\right]^3; -3 \le x_i \le 3; \quad i = 1,\ldots,n \tag{18}$$

Global optimum value [26]: 0

4.5.5 20D SUR-T1-16

$$f(\boldsymbol{x}) = \sum_{i=1}^{5}\Big[(x_i + 10x_{i+5})^2 + 5(x_{i+10} - x_{i+15})^2 + (x_{i+5} - 2x_{i+10})^4 + 10(x_i - x_{i+15})^4\Big]; -2 \le x_i \le 5; i = 1, \ldots, 5 \tag{19}$$

Global optimum value [26]: 0

4.5.6 10D, 20D, and 30D Griewank Function (HDGR10, HDGR20, and HDGR30, Respectively)

$$f(x) = \sum_{i=1}^{n}\frac{x_i^2}{4000} - \prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1; -600 \le x_i \le 600 \tag{20}$$

Global optimum value: 0

Note that for testing with DIRECT, a second set of results are obtained using skewed bounds ranging from −400 to 800 because DIRECT always evaluates the center point, which could otherwise bias the comparison.

4.5.7 10D, 20D, and 30D Zakharov Function (HDZF10, HDZF20, and HDZF30, Respectively)

$$f(\boldsymbol{x}) = \sum_{i=1}^{n}x_i^2 + \left(\sum_{i=1}^{n}0.5ix_i\right)^2 + \left(\sum_{i=1}^{n}0.5ix_i\right)^4; -5 \le x_i \le 10 \tag{21}$$

Global optimum value: 0

4.5.8 Engineering Application: 30D Cantilever Problem (HDCP30). The chosen engineering application problem involves minimizing the tip deflection of a stepped cantilever beam subject to constraints, as shown in Fig. 1. This problem is a modified version of the problem included in Refs. [27] and [28], which was first proposed in Ref. [29]. As the problem is complex, there is no known analytical optimum.

For testing the proposed algorithm, a cantilever beam with $d = 10$ steps is chosen with a $P = 50\,\text{kN}$ force on the tip. $E = 200\,\text{GPa}$ and $\sigma_{\text{allow}} = 35 \times 10^7\,\text{Pa}$ are selected as the properties for the used material. For each beam step, there exists three variables for the optimization: width (b_i), height (h_i), and length (l_i) of the beam step. Therefore, 30 input variables exist in this minimization problem and are arrayed in the following order:

$$X = [b_1, h_1, l_1, b_2, h_2, l_2, \ldots, b_{10}, h_{10}, l_{10}] \tag{22}$$

The objective is to minimize the tip deflection (δ), expressed as the summation of all the deflections [30]

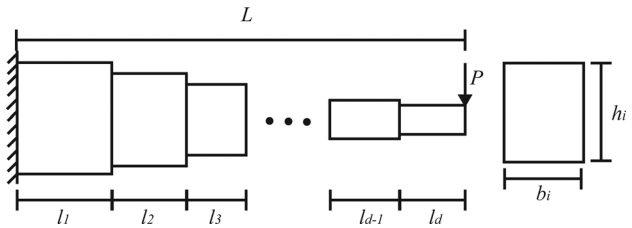

Fig. 1 Stepped cantilever beam with d steps

$$\begin{aligned}\delta &= \int_0^{l_d}\frac{Px_d^2}{EI_d}dx_d + \int_0^{l_{d-1}}\frac{P(x_{d-1} + l_d)^2}{\text{EI}_{d-1}}dx_{d-1} + \cdots \\ &\quad + \int_0^{l_1}\frac{P(x_1 + l_2 + l_3 + \cdots + l_d)^2}{\text{EI}_1} \\ &= \frac{Pl_d^3}{3E\left(\frac{b_d h_d^3}{12}\right)} + \frac{P}{3E\left(\frac{b_{d-1}h_{d-1}^3}{12}\right)}\left[(l_{d-1} + l_d)^3 - l_d^3\right] + \cdots \\ &\quad + \frac{P}{3E\left(\frac{b_1 d_1^3}{12}\right)}\left[(l_1 + l_2 + \cdots + l_d)^3 - (l_2 + l_3 + \cdots + l_d)^3\right] \\ &= \frac{P}{3E}\sum_{i=1}^{d}\left[\frac{12}{b_i h_i^3}\left(\left(\sum_{j=i}^{d}l_j\right)^3 - \left(\sum_{j=i+1}^{d}l_j\right)^3\right)\right]\end{aligned} \tag{23}$$

where P is the concentrated load, E is the material elasticity modulus, and I_i is the moment of inertia about the neutral axis. There are "$2 \times d + 1$" constraints in the problem, where d is the number of steps. First, the bending stress in all d steps should be less than the allowable bending stress (σ_{allow}). The constraints for each step beam can be described as

$$\frac{6P\sum_{j=i}^{d}l_j}{b_i h_i^2} \le \sigma_{\text{allow}} \quad i = 1, 2, \ldots, d \tag{24}$$

In addition, the aspect ratios of all cross sections make another set of d constraints that can be shown as

$$\frac{h_i}{b_i} \le \text{AR} \quad i = 1, 2, \ldots, d \tag{25}$$

where AR is the aspect ratio. The volume of the material used in the beam should have a maximum which make another constraint

$$\sum_{i=1}^{d}b_i h_i l_i \le V_{\max} \tag{26}$$

The last constraint is that the total length of the cantilever beam should be more than a specified value

$$\sum_{i=1}^{d}l_i \ge L_{\min} \tag{27}$$

$L_{\min}$ is the minimum required length that is expected for the beam. In brief, the minimization problem can be rewritten as

$$\delta = f(X) = \frac{P}{3E}\sum_{i=1}^{10}\left[\frac{12}{x_{3i-2}x_{3i-1}^2}\left(\left(\sum_{j=i}^{10}x_{3j}\right)^3 - \left(\sum_{j=i+1}^{10}x_{3j}\right)^3\right)\right] \tag{28}$$

Subject to

$$\begin{aligned}&\frac{6P}{x_{3i-2}x_{3i-1}^2}\sum_{j=i}^{10}x_{3j} \le \sigma_{\text{allow}} \\ &\frac{x_{3i-1}}{x_{3i-2}} \le \text{AR} \quad i = 1, 2, \ldots, 10 \\ &\sum_{j=1}^{10}x_{3j-2}x_{3j-1}x_{3j} \le V_{\max}\end{aligned} \tag{29}$$

$$\sum_{j=1}^{10} x_{3j} \geq L_{\min}$$

$$i = 1, 2, \ldots, 10$$

AR = 25, $V_{\max} = 1.2$, and $L_{\min} = 5.1$ m are selected as the aspect ratio and minimum length for the beam. The following bounds are selected for the variables

$$\begin{aligned} &0.01\,\text{m} < b_i < 0.05\,\text{m} \\ &0.30\,\text{m} < h_i < 0.65\,\text{m} \\ &0.50\,\text{m} < l_i < 1.00\,\text{m} \\ &i = 1, 2, \ldots, 10 \end{aligned} \tag{30}$$

The authors would like to note that this problem is a 1D idealization based on beam theory, and does not factor in effects of stress concentrations at the interfaces between the steps. Also, this cantilever beam problem can be easily solved using fewer function evaluations using a gradient-based method such as those found in MATLAB's Optimization Toolbox, even if gradient data is not directly available (i.e., finite differencing is used to obtain gradient information). Thus, this problem is used only for illustrative and benchmarking purposes.

4.6 Simulation Results

4.6.1 First Set of Simulations (MPS, TRMPS2, GA, DIRECT). Results from the first set of simulations are shown in Table 1, where nfe is the average number of function evaluations over 30 runs; fval is the average optimum function values; and STD indicates the standard deviation of optimum function values. Table 1 indicates that on most problems TRMPS2 outperforms GA and DIRECT, and that all three aforementioned algorithms outperform MPS by a very large margin.

In the F16 problem, TRMPS2 produced very tight results that are close to the optimum and exhibit little variation and that GA generates surprisingly poor results that have relatively large variation despite using more function evaluations than TRMPS2 and MPS. In the Rosenbrock problem, TRMPS2 produces the best and most consistent results on average, followed by GA and then DIRECT while MPS performs poorly. MPS produces large variations in optimum function values due to its much higher reliance on random sampling compared to the other algorithms tested. Note that since DIRECT is a deterministic algorithm, it generates the same results every run and thus exhibits no variance in results. Many of the figures have MPS removed, because MPS performs extremely poorly with the increased dimensionality of the problem and skews the box plot. This is a strong indication that MPS does not handle high dimensionality well. TRMPS2, DIRECT,

Table 1 First set simulation results (30 problems, 5000 max nfe)

		MPS			TRMPS2		
Problem	Optimum fval	Mean nfe	Mean fval	STD of fval	Mean nfe	Mean fval	STD of fval
F16	25.875	2443	29.07	28.9	561.2	26.12	0.315
10D Rosenbrock	0	4902	229.5	177.8	3828.33	5.548	1.99
20D Rosenbrock	0	4891	3.315×10^4	3.49×10^4	5000	17.98	1.41
30D Rosenbrock	0	5000	9.45×10^4	9.65×10^4	5000	21.53	7.54
10D SUR-T1-14	0	4721	20.11	15.1	5000	1.086	0.201
20D SUR-T1-14	0	5000	6765	6476	5000	2.192	0.848
30D SUR-T1-14	0	5000	3.42×10^4	3.38×10^4	5000	3.536	1.536
10D PUR-T1-13	0	4655	4.237×10^7	6.675×10^5	4153	9.84	26.2
20D PUR-T1-13	0	5000	2.182×10^{13}	1.572×10^{13}	5000	8.399×10^5	1.45×10^6
30D PUR-T1-13	0	5000	7.18×10^{15}	6.345×10^{15}	5000	3.189×10^8	3.55×10^8
20D SUR-T1-16	0	5000	1.34×10^4	1336	5000	0.745	0.362
10D Griewank	0	140	0.184	0.042	2352	0.1328	0.0901
20D Griewank	0	5000	205.9	209.44	5000	0.138	0.0370
30D Griewank	0	5000	396.6	405.20	5000	0.252	0.0506
10D Zakharov	0	4914	2.73×10^6	1.65×10^6	3532	0.371	0.764
20D Zakharov	0	5000	2.36×10^{11}	2.31×10^{11}	5000	0.235	0.1888
30D Zakharov	0	5000	1.112×10^{13}	1.180×10^{13}	5000	31.03	14.71
30D Cantilever	—	5000	0.0320	0.0316	990	0.0153	2.12×10^{-3}

		GA			Direct	
Problem	Optimum fval	Mean nfe	Mean fval	STD of fval	nfe	fval
F16	25.875	5000	45.3	4.182	5355	26.37
10D Rosenbrock	0	5000	6.30	1.3931	5139	7.85
20D Rosenbrock	0	5000	44.73	19.915	5331	17.89
30D Rosenbrock	0	5000	99.83	24.21	5605	27.9
10D SUR-T1-14	0	5000	1.418	0.3853	5151	1.36
20D SUR-T1-14	0	5000	31.35	24.91	5245	5.28
30D SUR-T1-14	0	5000	243.6	130.4	5149	252
10D PUR-T1-13	0	5000	9.979	23.447	23903	9.75×10^9
20D PUR-T1-13	0	5000	7.97×10^5	9.71×10^5	8919	8.58×10^{13}
30D PUR-T1-13	0	5000	7.510×10^8	7.152×10^8	8071	1.01×10^{16}
20D SUR-T1-16	0	5000	1.169	0.6658922	5381	152.4
10D Griewank	0	5000	9.56×10^{-4}	5.614×10^{-4}	5237 22,983	0.00 92.1 (right number, skewed bounds)
20D Griewank	0	5000	$5.66e \times 10^{-3}$	2.077×10^{-3}	5333 7175	0.00 201 (right number, skewed bounds)
30D Griewank	0	5000	0.0125	3.26×10^{-3}	5349 6857	0.00 301 (right number, skewed bounds)
10D Zakharov	0	5000	1.83	1.73	5139	34.99
20D Zakharov	0	5000	28.6	122.88505	5175	109
30D Zakharov	0	5000	8.067	2.776	5317	179.6
30D Cantilever	No known optimum	1290 5000	0.120 0.0120	0.04526	5383	0.02905

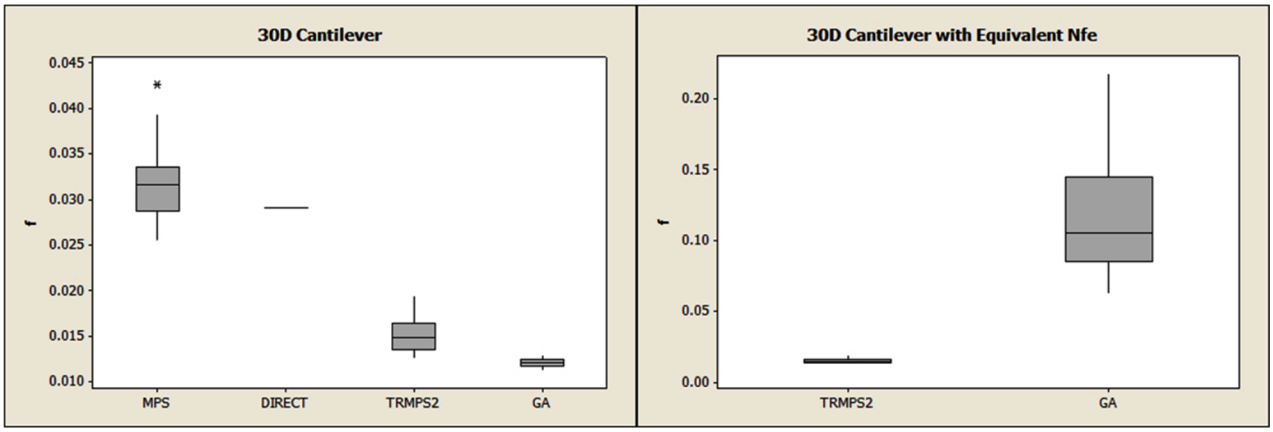

Fig. 2 (*a*) 30D cantilever with 5000 points and (*b*) 30D cantilever problem with equivalent nfe

and GA, by contrast, adapt reasonably well to the increase in dimensionality. The results of the SUR-T1-14 and SUR-T1-16 problems are similar to that of the Rosenbrock problem.

In the PUR-T1-13 problem the performance gap between TRMPS2 and GA is less dramatic than that of previous problems while DIRECT and MPS perform poorly. Table 1 shows that MPS, TRMPS2, and GA all exhibit large standard deviation that increase dramatically with greater dimensionality, which suggests that the PUR-T1-13 problem is very sensitive to minor variations in input parameters. Even then, TRMPS2 generates fewer outliers and a tighter band of solutions than GA.

In the Griewank problem TRMPS2 performs worse than GA and DIRECT. However, the performance of DIRECT cannot be attributed to its superiority since DIRECT always evaluates the center point, which happens to be the global optimum for the problem [8]. Skewing the bounds of the problem so that the bounds are from −400 to 800 and testing with DIRECT yields much worse results, as shown in Table 1. While GA performs better than TRMPS2, in practice the difference in quality is slight given that a randomly generated sample point in the search space can have a function value greater than 800 for the 20 variable problem and 1300 for the 30 variable problem while the differences in the optimal values for the two algorithms are in the scale of 0.1. In the Zakharov problem, the performance of TRMPS2 and GA are again comparable, with TRMPS2 generating more accurate solutions than GA in 10D and 20D Zakharov problems, as shown in Table 1.

Finally, in the 30 variable cantilever design engineering application problem TRMPS2 ultimately outperforms MPS, DIRECT, and GA. Although Fig. 2(*a*) shows that GA generates the best solutions, GA uses 5000 function evaluations while TRMPS2 uses much less and converges earlier. The maximum number of generations is lowered in another set of runs with GA to obtain close to an average of 1300 function evaluations in order to produce a better comparison with TRMPS2, with the results in Fig. 2(*b*) showing that TRMPS2 generates better optimums with greater consistency than GA when the numbers of function evaluations are comparable.

While the performance of TRMPS2 is satisfactory in the results presented, this comes at the expense of parallelization as TRMPS2 can sample much fewer points per iteration than GA. However, in practice computing resources and software licenses are often constrained and TRMPS2 is useful in such conditions.

4.6.2 Second Set of Simulations (EGO, TRMPS2). The second set of simulations is necessitated by EGO's high computational expense. More than a month of testing on several computers still did not yield anywhere close to the number of data points required in the first set of tests. Thus, the second set of simulations is used to benchmark the EGO algorithm in reasonable time and to illustrate its computational cost. The run parameters have been set, as shown in Table 2, to make EGO computationally viable for our second set of tests. We have made every effort to ensure the settings used have minimal impact on the performance of the algorithm. EGO's default setting is to add one point to the model per iteration, which is extremely expensive for high numbers of function evaluations. Thus, all runs are limited to a maximum of 1000 function evaluations to reduce computational expense. We have kept the number of points generated per cycle to one in order to maximize the performance of the algorithm. Also, the number of initial expensive samples scales with the dimensionality of the problem. As mentioned in Ref. [7], the initial samples should be approximately 10**nv*. The interpoint spacing should also be a round number and can be calculated via $1/(nv-1)$. Based on these requirements, the numbers of initial samples are 101, 201, and 301 for 10, 20, and 30 variable problems, respectively. The maximum number of cycles for each problem has been adjusted so that there is a maximum of 1000 evaluations per run.

The results show that TRMPS2 performs better than EGO on all problems tested by large margins, but because of EGO's computational cost, too few test functions are run and thus a definitive conclusion cannot be drawn.

Table 3 shows the computational cost of both EGO and TRMPS2. Both algorithms are run on the same computer under the same conditions with only one run performed at a time, and the computer has a first generation Intel i7 quad-core CPU with 16 GB of memory. These measures allow for a fair comparison of the computational cost of the algorithms. For the test problems, the cost of function evaluation is negligible. Therefore, the time spent for testing is mostly caused by running the algorithms. Table 3 shows that computational cost for both algorithms scale with the dimensionality of the problem. However, TRMPS2's run times are fairly fast, with the longest average run time being close to seven minutes with the 30D Zakharov problem. Dividing TRMPS2's total time by the number of function evaluations, it is equivalent to 0.41 s of algorithm CPU time per function evaluation, which is negligible compared to a reasonably expensive black box problem taking several minutes per evaluation. EGO's run times are much slower, with the shortest average run time being 10.3 h. This is equivalent to 0.62 min of algorithm CPU

Table 2 EGO run settings

Number of variables	Max points per cycle	Initial samples	Max number of cycles
10	1	101	899
20		201	799
30		301	699

Table 3 Second set simulation results (3 problems, 1000 max nfe)

		EGO				TRMPS2			
Problem	Optimum fval	Mean nfe	Mean fval	STD of fval	Mean run time (s)	Mean nfe	Mean fval	STD of fval	Mean run time (s)
10D Rosenbrock	0	1000	453.22	182.38	3.70×10^4 (10.3 h)	1000	23.18	24.81	85.96
20D Griewank	0	1000	237.18	19.1	4.56×10^5 (126.7 h)	1000	1.27	0.211	198.3
30D Zakharov	0	1000	2.6643×10^{13}	5.987×10^{12}	2.86×10^5 (79.4 h)	1000	38.23	57.6	414.3

time per function evaluation, which is not negligible. The longest average run time is 126.7 h, which is equivalent to 7.6 min per function evaluation. Thus, the results indicate that EGO may have nontrivial computational cost from the algorithm itself relative to some practical black box problems and may not reduce overall optimization cost under those circumstances, even if the number of function evaluations is significantly reduced relative to other optimization algorithms.

5 Final Remarks

This work proposed a TRMPS2 optimization algorithm to improve optimization performance for high dimensional design problems. The method improves on the original MPS optimization algorithm by introducing a new trust region approach for guided sampling of the search space using two trust regions. The approach involves incorporating a metamodeling and MPS process within each trust region. The trust region approach has also been augmented by the addition of the LFV strategy and a customized local optimization method, which makes more efficient use of limited metamodeling resources in the high dimensional space.

The results show promises as compared to the original MPS method as well as the standard GA, DIRECT, and EGO. For high dimensional problems, except for problems of simple variable structures, it seems that building metamodels in a subregion to compensate for the limit on metamodel size is a promising approach. The simulation results indicate that TRMPS2 is significantly better than MPS for operating in the high dimensional space. The simulation results also indicate that, overall, TRMPS2 generates more accurate solutions more consistently than standard GA when both algorithms are applied to HEB problems. The results also show that TRMPS2 generates better solutions when compared with DIRECT, and has significantly less computational cost when compared to EGO. The results suggest that TRMPS2 is overall a promising global optimization approach for HEB problems.

It is also found through the testing that TRMPS2's performance drops when it is difficult to get feasible points for highly constrained problems especially those with equality constraints. Future work will be to develop an advanced constraint handling strategy for TRMPS2.

Acknowledgment

The authors are grateful for the financial support of the National Sciences and Engineering Research Council (NSERC), without which this work would not have been possible.

References

[1] Haftka, R., Scott, E., and Cruz, J., 1998, "Optimization and Experiments: A Survey," ASME Appl. Mech. Rev., **51**(7), pp. 435–448.
[2] Gu, L., 2001, "A Comparison of Polynomial Based Regression Models in Vehicle Safety Analysis," ASME Paper No. DAC-21063.
[3] Duddeck, F., 2008, "Multidisciplinary Optimization of Car Bodies," Struct. Multidiscip. Optim., **35**(4), pp. 375–389.
[4] Mitchell, M., 1996, *An Introduction to Genetic Algorithms*, MIT, Cambridge, MA.
[5] Bertsimas, D., and Tsitsiklis, J., 1993, "Simulated Annealing," Stat. Sci., **8**(1), pp. 10–15.
[6] Poli, R., Kennedy, J., and Blackwell, T., 2007, "Particle Swarm Optimization: An Overview," Swarm Intell., **1**(1), pp. 33–57.
[7] Jones, D. R., Schonlau, M., and Welch, W. J., 1998, "Efficient Global Optimization of Expensive Black-Box Functions," J. Global Optim., **13**(4), pp. 455–492.
[8] Jones, D., Perttunen, C., and Stuckman, B., 1993, "Lipschitzian Optimization Without the Lipschitz Constant," J. Optim. Theory Appl., **79**(1), pp. 157–181.
[9] Shan, S., and Wang, G. G., 2010, "Metamodeling for High Dimensional Simulation-Based Design Problems," ASME J. Mech. Des., **132**(5), pp. 1–11.
[10] Zhan, Z., Zhang, J., Li, Y., and Chung, H., 2009, "Adaptive Particle Swarm Optimization," IEEE Trans. Syst., Man, Cybern., Part B, **39**(6), pp. 1362–1381.
[11] Wang, L., Shan, S., and Wang, G., 2004, "Mode-Pursuing Sampling Method for Global Optimization on Expensive Black-Box Functions," J. Eng. Optim., **36**(4), pp. 419–438.
[12] Sharif, B., Wang, G. G., and ElMekkawy, T. Y., 2008, "Mode Pursuing Sampling Method for Discrete Variable Optimization on Expensive Black-Box Functions," ASME J. Mech. Des., **130**(2), p. 021402.
[13] Duan, X., Wang, G., Kang, X., Niu, Q., Naterer, G., and Peng, Q., 2009, "Performance Study of Mode-Pursuing Sampling Method," J. Eng. Optim., **41**(1), pp. 1–21.
[14] Conn, A. R., Gould, N. I. M., and Toint, P. L., 2000, *Trust Region Methods*, J. E. Dennis, ed., MPS-SIAM, Philadelphia, PA.
[15] Addis, B., and Leyffer, S., 2006, "A Trust Region Algorithm for Global Optimization," Comput. Optim. Appl., **35**(3), pp. 287–304.
[16] Alexandrov, N., Dennis, J., Lewis, R., and Torczon, V., 1997, "A Trust Region Framework for Managing the Use of Approximation Models in Optimization," Struct. Optim., **15**(1), pp. 16–23.
[17] Rodríguez, J., Watson, L., and Renaud, J. E., 1998, "Trust Region Augmented Lagrangian Methods for Sequential Response Surface Approximation and Optimization," ASME Design Engineering Technical Conferences, Vol. **120**(1), pp. 58–66.
[18] Cheng, G., and Wang, G., 2012, "Trust Region Based MPS Method for Global Optimization of High Dimensional Problems," AIAA Paper No. 2012-1590.
[19] Fu, J., and Wang, L., 2002, "A Random-Discretization Based Monte Carlo Sampling Method and Its Applications," Methodol. Comput. Appl. Probab., **4**(1), pp. 5–25.
[20] SIMULIA, 2009, "Isight: Automate Design Exploration and Optimization." Available at: http://www.simulia.com/download/products/Isight_Brochure_FINAL.pdf, Accessed 2014.
[21] Red Cedar Technology, 2014, "SHERPA—An Efficient and Robust Optimization/Search Algorithm." Available at: http://www.redcedartech.com/pdfs/SHERPA.pdf, Accessed 2014.
[22] Houck, C., Joines, J., and Kay, M., 1995, "A Genetic Algorithm for Function Optimization: A MATLAB Implementation," ACM Trans. Math. Software.
[23] Viana, F., 2013, "SURROGATES Toolbox," https://sites.google.com/site/felipeacviana/surrogatestoolbox, Accessed 2013.
[24] Jones, D., 2001, "The Direct Global Optimization Algorithm," Encycl. Optim., **1**, pp. 431–440.
[25] Finkel, D., 2004, "Research and Codes." Available at: http://www4.ncsu.edu/~ctk/Finkel_Direct/, Accessed 2013.
[26] Schittkowski, K., 1987, *More Test Examples for Nonlinear Programming Codes* (Lecture Notes in Economics and Mathematical Systems), Springer-Verlag, New York.
[27] Hsu, Y., Wang, S., and Yu, C., 2003, "A Sequential Approximation Method Using Neural Networks for Engineering Design Optimization Problems," Eng. Optim., **35**(5), pp. 489–511.
[28] Thanedar, P., 1995, "Survey of Discrete Variable Optimization for Structural Design," J. Struct. Eng., **121**(3), pp. 301–306.
[29] Haji Hajikolaei, K., Pirmoradi, Z., and Wang, G., "Decomposition for Large-Scale Global Optimization Based on Quantified Variable Correlations Uncovered by Metamodeling," J. Eng. Optim. (in press).
[30] Schutte, J., and Haftka, R., 2010, "Global Structural Optimization of a Stepped Cantilever Beam Using Quasi-Separable Decomposition," Eng. Optim., **42**(4), pp. 347–367.