Proceedings of the ASME 2010 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference
IDETC/CIE 2010
August 15-18, 2010, Montreal, Quebec, Canada

DETC2010-28355

CONSTRAINT IMPORTANCE MODE PURSUING SAMPLING FOR CONTINUOUS GLOBAL OPTIMIZATION

Moslem Kazemi *
School of Engineering Science
Simon Fraser University
Burnaby, BC Canada
E-mail: moslemk@cs.sfu.ca

G. Gary Wang
School of Engineering Science
Simon Fraser University
Burnaby, BC Canada
E-mail: gary_wang@sfu.ca

Shahryar Rahnamayan
Faculty of Engineering and Applied Science
University of Ontario Institute of Technology
Oshawa, ON Canada
E-mail: Shahryar.Rahnamayan@uoit.ca

Kamal Gupta
School of Engineering Science
Simon Fraser University
Burnaby, BC Canada
E-mail: kamal@cs.sfu.ca

ABSTRACT

Many engineering design problems deal with global optimization of constrained black-box problems which is usually computation-intensive. Ref. [1] proposed a Mode-Pursuing Sampling (MPS) method for global optimization based on a sampling technique which systematically generates more sample points in the neighborhood of the function mode while statistically covering the entire problem domain. In this paper, we propose a novel and more efficient sampling technique which greatly enhances the performance of the MPS method, especially in the presence of *expensive* constraints. The effective sampling of the search space is attained via biasing the sample points towards *feasible* regions and being away from the forbidden regions. This is achieved by utilizing the incrementally obtained information about the constraints, hence, it is called Constraint-importance Mode Pursuing Sampling (CiMPS). According to intensive comparisons and experimental verifications, the new sampling technique is found to be more efficient in solving constrained optimization problems compared to the original MPS method. To the best of our knowledge, this is the first metamodel-based global optimization method that directly aims at reducing the number of function evaluations for both expensive objective functions and constraints.

*Address all correspondence to this author.

1 Introduction

Due to the wide use of computation intensive tools in engineering design, metamodeling has become a popular approach in recent years [2, 3]. Most researchers use metamodels as a surrogate for the expensive computer model in the optimization, directly or adaptively, for the reduction of computational costs. In these studies, the objective function of an optimization problem is assumed to be expensive. The constraints, either box or more complicated linear or nonlinear constraints, are usually assumed to be cheap ([4, 5], to name a few). A cheap function is interpreted as having CPU time for one function evaluation at least an order of magnitude lower than that for an expensive function. In a real design practice, one will face situations in which the constraints can be computational expensive. For example, for vehicle design, the objective may be to reduce the life-cycle costs and one of the constraints is its crashworthiness. The crashworthiness is likely evaluated through crash simulation, which is a widely known computationally expensive process. It is therefore worthwhile to study metamodeling-based optimization techniques for both expensive objective functions and constraints.

In the field of metamodeling-based design optimization, there is little work on expensive constraints. Ref. [6] studied constraint handling in the context of their Efficient Global Optimization (EGO) algorithm. The authors compared the penalty method and the one that multiplies the expected improvement (of the objective) by the probability that the point is feasible. It

was found that these two methods have distinct merits, depending on how strictly the constraints need to be satisfied. The work in [6] tried to avoid sampling in infeasible areas, which indirectly reduces the computational costs for constraints. Refs. [7], [8], and [9] used constraint programming with the assistance of metamodeling to reduce the search space. In these works, constraints are either expensive or inexpensive; and the goal is to bring the optimization into a confined smaller space. Constraint programming, though promising, needs careful tuning and brings extra difficulties to the designer [10]. In general the lack of study on expensive constraints is perhaps due to the following reasons. First, it is found that if constraints are also approximated by surrogates, the obtained constrained optimum, which often rests on the boundary of the feasible space, may be quite far from the actual optimum because of approximation errors in both constraint and objective functions [11]. While there are still challenges to build an accurate surrogate for the objective, the constraints are then assumed inexpensive for convenience as well as for a better paper focus. Secondly, it is also perhaps because researchers overlooked the challenge of expensive constraints, as the authors did before.

For constrained optimization in general, there are classic methods such as Lagrange multipliers, quadratic programming, steepest descent method, and penalty methods [12]. When the functions (both objective and constraints) are black-box, many of the classic methods are not applicable. Ref. [13] gave a comprehensive review of constraint handling techniques in evolutionary computing, in which the functions are also black-box. Besides many algorithm specific methods such as various chromosome representations and operators, the penalty methods are of special interests to the authors. They reviewed six types of penalty methods, i.e., static penalty, dynamic penalty, annealing penalty, adaptive penalty, co-evolutionary penalty, and death penalty. A recent article [14] has also been devoted to constraint handling for evolutionary optimization.

There is also a large quantity of literatures on reliability-based design optimization (RBDO) when the uncertainties of variables and probabilistic constraints are taken into consideration. In this area, researchers in recent years started to consider constraints, or performance functions, as being expensive. A recent assemble of papers in this area is in [15]. There exist distinctive differences in theory and methodology on constraint handling between RBDO and deterministic optimization which is the focus of [15]. Therefore we refrain from delving deep into this area.

The present work has been motivated from the application of the Mode Pursing Sampling (MPS) method, a global optimization method originally developed in [1] for continuous variable optimization problems which later on was extended to mixed variable problems [16]. Through testing its performance [17], it is found that MPS took an excessive amount of CPU time for constraint handling, even for inexpensive constraints. This in fact brings down its performance for relatively high dimensional black-box problems ($n > 10$; n is the number of design variables). Later on, the MPS method was applied for crashworthiness optimization where constraints were expensive; and the need for a technique to handle expensive constraint arose. This work thus aims to develop a constraint handling approach for optimization problems involving both expensive objective and constraint functions. As discussed before, directly using surrogates for both types of functions in optimization could yield erroneous results due to the metamodeling errors. New techniques are therefore needed. This work is based on the framework of the MPS method which does not rely on accurate metamodels but rather use metamodels as a sampling guide.

In Section 2, MPS will be briefly reviewed and its constraint handling strategy is explained. The proposed approach will be described in Section 3. Experimental verifications and comparison analysis will be presented in Section 4, and finally, the work will be concluded in Section 5.

2 Mode Pursuing Sampling Method: Review and Issues

The Mode Pursuing Sampling method [1] integrates the technique of meta-modeling and a novel discriminative sampling method proposed in [18] in an iterative scheme for global optimization of black-box problems. It generates more sample points in the neighborhood of the function mode and fewer points in other areas as guided by a special sampling guidance function. Moreover, by continuous sampling in the global search space it avoids trapping in local minima.

The MPS method for minimizing a black-box function $f : \mathbb{R}^n \to \mathbb{R}$ is given as a pseudo-code in Fig. 1. It takes the set of constraints $\mathscr{G}_f = \{g_k(x) \leq 0 | k = 1, \ldots, K\}$ and the problem domain $\mathscr{D}_f \subset \mathbb{R}^n$ as inputs and returns the global minimum of $f(x)$ as output in case of success. The algorithm can be summerized in four steps as follow:

Step 1 (Initial sampling, lines 2 and 3): A set of m sample points $X = \{x_i \in \mathscr{D}_f | g_k(x_i) \leq 0, k = 1, \ldots, K; i = 1, \ldots, m\}$ is generated randomly at line 2 using function `SampleWithConstraints()` in the feasible region of the problem domain $D_f \subset \mathbb{R}^n$ where m is an arbitrary integer that usually increases as the dimension of the problem domain, n, increases. These m points are called *expensive* points since their function values are evaluated by the black-box function $f(x)$ at line 3.

Step 2 (Function approximation, lines 6-9): A piecewise-linear spline $\widehat{f}(x)$ is then fitted (line 6) to the set of expensive points X as a meta-model of the actual black-box function:

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

such that $\widehat{f}(x_i) = f(x_i)$, for $i = 1, \ldots, m$, with constant α_i. Then a sampling guidance function $h(x) = c_0 - \widehat{f}(x)$ is defined (line 7) where c_0 is a constant such that $c_0 > \max(\widehat{f}(x))$. The guidance function can be viewed as a probability density function (up to a normalizing constant) whose modes are located at those x_i's where the function values are the lowest among $f(x_i)$'s. N (usually large) number of valid

```
input : f(x) black box function; G_f
        = {g_k(x) ≤ 0 | k = 1,...,K} set of constraints; D_f
        ⊂ R^n problem domain
output: x_min global minimum of f(x), or null in case of
        failure
1  begin
2      X ← SampleWithConstraints(m, G_f, D_f);
3      V ← Evaluate(X, f);
4      iter = 1;
5      while iter ≤ MAX_ITERATION do
6          f̂ ← LinearSpline(X, V);
7          h ← c_0 − f̂;                    // c_0 > Max(f̂)
8          X_N ← SampleWithConstraints(N, G_f,
           D_f);
9          V_N ← Evaluate(X_N, h);
10         x_mod ← Mode(X_N, V_N);
11         X_m ← SampleTowardMode(m, X_N, V_N);
12         V_m ← Evaluate(X_m, f);
13         X ← X ∪ X_m;
14         V ← V ∪ V_m;
15         q ← (n+1)(n+2)/2 + 1;
16         [X_q, V_q] ← NeighborSamples(q, x_mod, X,
           V);
17         if QuadraticRegression(X_q, V_q) is
           accurate then
               // Perform local optimization
18             [x_min, v_min] ← FMinCon(f, x_mod, G_f);
19             if x_min ∈ HyperCube(X_q) then
20                 Return x_min;  // Minimum found
21             end
22         end
23         X ← X ∪ x_min;
24         V ← V ∪ v_min;
25         iter = iter + 1;
26     end
27     Return null;          // Minimum not found
28 end
```

FIGURE 1: The Mode Pursuing Sampling (MPS) optimization algorithm

sample points are then randomly generated within the feasible region of the problem domain again by calling function `SampleWithConstraints()` at line 8. These points are called *cheap* points since their function values are evaluated by the linear guidance function $h(x)$, not the objective function, hence, their function values will be referred as *approximation* values[1].

Step 3 (Mode pursuing sampling, lines 10-12): A discriminative sampling technique (see [18]) is then employed (function `SampleTowardMode()` at line 11) to draw another set of m sample points from the set of the cheap points obtained in Step 2 according to the $h(x)$ (please see [1] for implementation details). By construction these sample points have the tendency to concentrate about the maximum (or mode) of $h(x)$, which corresponds to the minimum of $\widehat{f}(x)$.

Step 4 (Quadratic regression and local optimization, lines 15-22): The fourth step involves a quadratic regression in a sub-area around the current minimum of $\widehat{f}(x)$ (or mode of $h(x)$) according to the discriminative sampling in Step 3. If the approximation in the sub-area is sufficiently accurate, local optimization is performed in this sub-area to obtain the minimum, x_{min}. The x_{min} is returned as the global minimum of $f(x)$ if it is located inside the identified sub-area around the mode of $h(x)$. Otherwise, it is added to the set of expensive points and the algorithm restart from Step 2. The reader is referred to [1] for more details on implementation of the above steps.

In short, the MPS is an algorithm which uses discriminative sampling as its engine and has an intelligent mechanism to use the information from past iterations to lead the search toward the global optimal. At each iteration of the MPS two types of approximations are used: (1) approximation of the entire function by fitting the meta-model given in Eqn. (1) to all expensive points (line 6), and (2) quadratic regression around the attractive sub-areas (line 17). The first approximation uses a piecewise-linear spline as the meta-model because of its simplicity. One should note that the accuracy of the meta-model is not very critical here comparing to the cases where meta-models are used as surrogates since it is only used to guide the search toward the function mode. Nonetheless, the MPS method does not dictate the exclusive use of the linear functions, and other types of meta-models can be applied in lieu of the linear model. The accuracy of the quadratic regression (second approximation) around the attractive areas is increased at each iteration due to the discriminative sampling which generates more and more sample points around attractive regions.

2.1 MPS Limitations in Constrained Optimization Problems

Several simulations and design examples have shown the effectiveness, robustness, and applicability of the MPS method in both continuous [1] and discontinuous [16] domains. Some of the limitations of the MPS method were discussed in [1] and a study was conducted in [17] comparing the performance of the MPS with traditional global optimization methods such as the Genetic Algorithms in solving global optimization of both expensive and inexpensive objective functions.

One of the main issues which, unfortunately, has been overlooked is the performance of the MPS method in presence of *expensive* constraints. In the examples provided in the early works mentioned above the cost of checking the constraints when generating random sample points (specifically in function `SampleWithConstraints()` called at line 8 of Fig. 1) is not considered in the total cost of the optimization. Due to the large number of constraint checks that the

[1]Through out this paper, *cheap* samples refer to those points evaluated by a spline approximation of the objective function, while *expensive* samples denote the points evaluated by the objective function itself which is usually more time consuming.

```
input : n, number of samples to generate; 𝒢, set of
        constraints; 𝒟, problem domain
output: X, set of n valid samples
1  begin
2      X = {};
3      i = 1;
4      while i <= n do
5          x ← rand(𝒟);
6          if CheckForConstraints(x, 𝒢) ==
           success then
7              X ← X ∪ {x};
8              i = i + 1;
9          end
10     end
11     Return X;
12 end
```

FIGURE 2: Function `SampleWithConstraints`($n, \mathscr{C}, \mathscr{D}$)

MPS method relies on, this cost could be a determinant factor for constrained optimization problems especially when the design problem consists of expensive constraints. This can be explained by taking a closer look at the strategy that MPS algorithm utilizes to generate sample points, in particular in function `SampleWithConstraints()` (given in Fig. 2) in which a large number of *cheap* random points need to be generated within the *feasible* region of the problem domain. Referring to the pseudo-code in Fig. 2, to make sure that each sample point falls into the feasible region of the problem domain, it is checked against the set of all the constraints using function `CheckForConstraints()` at line 6). If the sample point satisfies all the constraints then it is added to the set of valid samples. Otherwise, it is discarded and a new sample is randomly generated and checked for the constraints. The generation of random samples continues based on the above scheme until the required number of valid sample points are obtained.

Apparently, in constrained optimization problems the above strategy might result in a large number of constraint checks considering the relative size of the forbidden and feasible regions. Moreover, in the above strategy, the invalid samples are discarded and the information obtained through the constraint check is not used in the overall MPS optimization algorithm.

To overcome the above shortcomings of the MPS technique, we present a novel sampling technique which systematically biases the generation of sample points towards feasible regions of the problem domain using the information which is incrementally obtained about the constraints, hence, the name *Constraint-importance Mode Pursuing Sampling* (CiMPS). The proposed sampling technique results in a substantially less number of constraint checks and, hence, superior performance in solving constrained optimization problems comparing to the original MPS method. One should note that in the CiMPS method, the generation of sample points is still biased towards the function mode similar to the original MPS technique while we use the information obtained through invalid samples to guide the sampling towards feasible regions away from constraints. The proposed sampling strategy is explained in more details in the next section.

3 Constraint-importance Mode Pursuing Sampling (CiMPS)

The crux of our proposed sampling method, called Constraint-importance Mode Pursuing Sampling (CiMPS), is to utilize the information obtained about the constrains to bias the generation of samples towards feasible region of the problem domain, hence, resulting in a more efficient sampling of the space and substantially less number of constraint checks comparing to the mode pursuing sampling technique proposed in [1]. It is worthwhile to mention that the CiMPS technique still benefits from the advantages of original MPS technique by biasing the samples towards the mode of the objective function and, hence, the number of objective function evaluation is kept relatively low due to fast convergence of the optimization towards the mode of the function. Thus, in summary the CiMPS method provides efficient sampling of the problem domain by accounting for the information incrementally obtained about both the objective function and constraints. This strategy is shown to result in a substantially low number of both function evaluations and constraint checks as we explain next.

The CiMPS algorithm (given as a pseudo-code in Fig. 3) generally follows the four steps similar to the MPS algorithm explained in Section 2. Similar to the original MPS optimization method, to bias the sample points towards the mode of the objective function, the sampling technique proposed in [18] is employed as the core of the discriminative sampling in CiMPS method (via function `SampleTowardMode()` at line 11) where a set of m sample points is systematically selected from a large number of cheap points biased toward the function mode according to their guidance function values.

The CiMPS method incorporates two important modifications to improve the performance of the MPS method in the presence of expensive constraints, specifically at lines 8 and 12: the former relaxes the constraint check criterion when generating a large number of cheap sample points, hence, it results in substantially less number of constraint checks overall. The latter evaluates the function values of expensive sample points according to the result of constraint check for each sample to incorporate the information obtained about the constraints into the objective function approximation. This yields a more efficient sampling of the search space. These two improvements are detailed in the following sections.

3.1 Relaxing Constraint Checks

As we mentioned earlier in Section 2, in the MPS method a large number of feasible cheap sample points are generated at line 8 by calling function `SampleWithConstraints()` in which each sample point is checked against all the constraints and in case of no constraint violation, it is added to the set of cheap sample points. In the CiMPS algorithm, this condition is relaxed at line 8 by calling function `Sample()` in which the cheap sample points are generated randomly within the prob-

```
input : f(x) black box function; G_f
        = {g_k(x) ≤ 0 | k = 1,...,q} set of constraints; D_f
        ⊂ R^n problem domain
output: x_min global minimum of f(x), or null in case of
        failure
1  begin
2      X ← SampleWithConstraints(m, G_f, D_f);
3      V ← Evaluate(X, f);
4      iter = 1;
5      while iter ≤ MAX_ITERATION do
6          f̂ ← LinearSpline(X, V);
7          h ← c_0 − f̂;                    // c_0 > Max(f̂)
8          X_N ← Sample(N, D_f);
9          V_N ← Evaluate(X_N, h);
10         x_mod ← Mode(X_N, V_N);
11         X_m ← SampleTowardMode(m, X_N, V_N);
12         V_m ← EvaluateWithConstraints(m, X_m,
           f, G_f);
13         X ← X ∪ X_m;
14         V ← V ∪ V_m;
15         q ← (n+1)(n+2)/2 + 1;
16         [X_q, V_q] ← NeighborSamples(q, x_mod, X,
           V);
17         if QuadraticRegression(X_q, V_q) is
           accurate then
               // Perform local optimization
18             [x_min, v_min] ← FMinCon(f, x_mod, G_f);
19             if x_min ∈ hyper_cube(X_q) then
20                 Return x_min ;   // Minimum found
21             end
22         end
23         X ← X ∪ x_min;
24         V ← V ∪ v_min;
25         iter = iter +1;
26     end
27     Return null ;          // Minimum not found
28 end
```

FIGURE 3: Constraint-importance Mode Pursuing Sampling (CiMPS) algorithm

lem domain without being checked against the constraints (see pseudo-code in Fig. 4). Hence, some of the samples might fall into forbidden regions defined by the constraints. Please note that the objective function may be undefined in these regions, however, the cheap sample points are supposed to be evaluated by the guidance function $h(x)$ (line 9, Fig. 3) which is a linear spline approximation of the objective function and is defined everywhere in the problem domain even in the forbidden regions. Nonetheless, the function values obtained for these infeasible samples do not properly represent the underlying objective function. If they are not treated appropriately they would result in improper sampling of the forbidden regions and eventually yields an invalid global minimum. As we see in the next section, in our proposed

```
input : n, number of samples to generate; D, problem
        domain
output: X, set of n valid samples
1  begin
2      X = {};
3      for i = 1 to n do
4          x ← rand(D);
5          X ← X ∪ {x};
6      end
7      Return X;
8  end
```

FIGURE 4: Function `Sample`(n, $\mathscr{D}$)

sampling technique the information obtained through these invalid samples are utilized to bias the sampling away from the constraints and towards the feasible regions of the problem domain.

The set of cheap sample points generated as explained above is then sampled by function `SampleTowardMode()` at line 11 using the sampling technique in [18] to obtain a set of m *expensive* sample points which are biased towards the mode of the guidance function. This step follows the implementation of the MPS technique proposed in [1].

3.2 Constraint-importance Sampling

The expensive sample points obtained using function `SampleTowardMode()` are used next to perform a quadratic regression (line 22) around the function mode, and later on are added to the set of expensive sample points (lines 23 and 24) for further improvement of the spline approximation of the black-box function at line 6. Therefore, to provide more accurate approximations, these samples should be evaluated by the objective function to obtain their exact function values. However, as mentioned above, some of these samples might fall into forbidden regions where the objective function may be undefined.

Therefore, these sample points are especially treated at line 12 by calling function `EvaluateWithConstraints()` in which each sample is first checked against all the constraints (see pseudo-code in Fig. 5). If a sample point satisfies all constraints then its actual function value is evaluated by the objective function. Otherwise, an appropriate penalty value is assigned as its function value (see line 7, Fig. 5). Two schemes are proposed to choose the appropriate penalty for invalid sample points:

Fixed Penalty If the user has some information regarding the maximum value that the objective function can achieve over the problem domain, then a value equal to or greater than the maximum can be selected as a fixed penalty for all infeasible samples. This information can be obtained by the user at the beginning by examining the objective function.

Variable Penalty The second scheme consists of iteratively modifying the penalty value as follows. At each iteration, the penalty value would be set equal to or greater than the maximum function value of the valid samples which have been already identified up to the current iteration. This ap-

```
input  : n, number of points to be evaluated; X, set of n
         sample points to be evaluated; f, the objective
         function to evaluate X; 𝒢, set of constraints
output: V, value functions of points in X, i.e.
         V = {v_i = f(x_i) | x_i ∈ X, i = 1,...,n}
1  begin
2      V = {};
3      foreach x in the set X do
4          if CheckForConstraints(x, 𝒢) ==
           success then
5              v ← f(x);
6          else
7              v ← A_PENALTY_VALUE;
8          end
9          V ← V ∪ {v};
10     end
11     Return V;
12 end
```

FIGURE 5: Function `EvaluateWithConstraints`(n, X, f, $\mathscr{C}$)

proach is desirable and less demanding since usually global information about the black-box function is not available beforehand and the fixed penalty scheme may not be applicable.

Interestingly, assigning an appropriate penalty obtained using either of above two schemes imposes a large function value to the approximated objective function $\widehat{f}(x)$ in forbidden regions represented by infeasible samples. This in turn results in low values for the guidance function $h(x)$ (in forbidden region), hence, the generation of sample points in function `SampleTowardMode()` will be biased away from the forbidden regions because, according to the sampling technique in [18], function `SampleTowardMode()` generates more sample points in the neighborhood of the function mode (regions with high fitness values) and generates less sample points in areas with low fitness.

4 Simulation Results and Design Examples

In this section, the effectiveness of the CiMPS method is evaluated on a number of benchmark functions (Section 4.1) and design examples (Section 4.2), and its performance is compared with the original MPS method and some previous works.

4.1 Benchmark Functions

The CiMPS method has been tested and compared with the orginal MPS method on a number of constrained optimization benchmarks, including:

QF A simple quadratic function $f_{QF} : [-3,3]^2 \to \mathbb{R}$

$$f_{QF}(x) = f_{QF}(x_1,x_2) = (x_1+1)^2 + (x_2-1)^2 \qquad (2)$$

subject to

$$\begin{aligned} g_1(x) &= 1-(x_1-1)^2-(x_2-1)^2 \le 0, \\ g_2(x) &= 2.25-x_1^2-(x_2+1.5)^2 \le 0, \end{aligned} \qquad (3)$$

with a global mimimum function value of $\min f_{QF} = 0.0$ at $x_{min} = (-1,1)$.

SC Six-hump camel back function $f_{SC} : [-2,2]^2 \to \mathbb{R}$

$$f_{SC}(x) = f_{SC}(x_1,x_2) = 4x_1^2 - \frac{21}{10}x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4 \qquad (4)$$

subject to

$$\begin{aligned} g_1(x) &= 0.25-(x_1-1)^2-(x_2-1)^2 \le 0, \\ g_2(x) &= 0.25-(x_1-1)^2-(x_2+1)^2 \le 0, \\ g_3(x) &= 0.25-(x_1+1)^2-(x_2-1)^2 \le 0, \\ g_4(x) &= 1.0-(x_1+1)^2-(x_2+1)^2 \le 0. \end{aligned} \qquad (5)$$

The $f_{SC}(x)$ has six local minima of which two (at $x = (-0.090, 0.713)$ and $(-0.090, -0.713)$) are global minima with equal function value of $\min f_{SC}(x) = -1.032$.

GP Goldstien-Price function $f_{GP} : [-2,2]^2 \to \mathbb{R}$

$$\begin{aligned} f_{GP}(x) &= f_{GP}(x_1,x_2) = \\ &\big(1+(x_1+x_2+1)^2(19-14x_1+3x_1^2-14x_2+6x_1x_2+3x_2^2)\big) \\ &\big(30+(2x_1-3x_2)^2(18-32x_1+12x_1^2+48x_2-36x_1x_2+27x_2^2)\big) \end{aligned} \qquad (6)$$

subject to

$$\begin{aligned} g_1(x) &= 0.5^2-(x_1-1)^2-(x_2-1)^2 \le 0, \\ g_2(x) &= 0.5^2-(x_1-1)^2-(x_2+1)^2 \le 0, \\ g_3(x) &= 0.5^2-(x_1+1)^2-(x_2-1)^2 \le 0, \\ g_4(x) &= 0.5^2-(x_1+1)^2-(x_2+1)^2 \le 0, \end{aligned} \qquad (7)$$

with a global minimum function value of $\min f_{GP}(x) = 3.0$ at $x_{min} = (0,-1)$.

HN Hartman function $f_{HN} : [0,1]^6 \to \mathbb{R}$

$$f_{HN}(x) = -\sum_{i=1}^{4} c_i \exp\Big[-\sum_{j=1}^{n} \alpha_{ij}(x_j - p_{ij})^2\Big] \qquad (8)$$

where α_{ij} and p_{ij} can be found in [1], subject to

$$\begin{aligned} g_1(x) &= x_1-0.5 \le 0, \\ g_2(x) &= x_2-0.5 \le 0, \end{aligned} \qquad (9)$$

with a global minimum function value of $\min f_{HN}(x) = -3.322$.

FIGURE 6: A tension/compression coil spring

F16 A function of 16 variables $f_{F16} : [-1,0]^{16} \rightarrow \mathbb{R}$

$$f_{F16}(x) = \sum_{i=1}^{16}\sum_{j=1}^{16} a_{ij}(x_i^2 + x_i + 1)(x_j^2 + x_j + 1) \qquad (10)$$

where a_{ij} for $i, j = 1, \ldots, 16$ is given in [1], subject to

$$\begin{aligned} g_1(x) &= x_1 + 0.2 \leq 0, \\ g_2(x) &= x_2 + 0.2 \leq 0, \\ g_3(x) &= x_{10} + 0.2 \leq 0, \\ g_4(x) &= x_{11} + 0.1 \leq 0, \end{aligned} \qquad (11)$$

with a global minimum function value of $\min f_{HN}(x) = 25.875$ at $x_{min} = (-0.5, -0.5, \ldots, -0.5)$.

Information related to each of the above problems and the run settings has been summarized in Tab. 1. For each problem, 30 runs have been carried out using the MPS and CiMPS methods and their performances are compared based on two cost indicators: (1) number of objective function evaluations (nfe), and (2) number of constraint checks (ncc). Both CiMPS and MPS methods share the same run settings: N, the number of cheap points generated at each iteration, the number of contours by which the cheap points are grouped, and m the number of expensive sample points generated at each iteration (the quadratic fitting accuracy R^2 is set to 0.999 for all test problems). For a detailed explanation of these settings and their effects one can refer to [1].

The simulation results obtained using the MPS and CiMPS methods for above problems are summarized in Tab. 2. As it is shown, for all test examples, the number of constraint checks (ncc) by the CiMPS method is significantly lower compared to the corresponding number of constraint checks by the MPS method. Moreover, except for QF, on all other functions CiMPS performs better in term of nfe.

4.2 Design Examples with Expensive Constraints

Two well known engineering design problems are used to evaluate the performance of the proposed method: (1) Minimization of Weight of the Spring, and (2) Pressure Vessel Design. Both are constrained optimization problems consisting several expensive constraints. Each problem is described with its corresponding constraints, bounds, and objective function in the following sections.

4.2.1 Minimization of Weight of the Spring This problem has been used as a test benchmark in literature, e.g. [12, 19–21]. In its standard form [12], it consists of designing a tension/compression spring (shown in Fig. 6) to carry a given axial load. The objective is to minimize the weight of the spring f_{WS} as

$$f_{WS}(x_1, x_2, x_3) = (x_3 + 2)x_2x_1^2 \qquad (12)$$

where $x_1 = d$ is the wire diameter, $x_2 = D$ is the mean coil diameter, and $x_3 = N$ denotes number of active coils, subject to the following constraints

$$\begin{aligned} g_1(x) &= 1.0 - \frac{x_2^3 x_3}{71875x_1^4} \leq 0 \quad \text{deflection constraint} \\ g_2(x) &= \frac{x_2(4x_2 - x_1)}{12566x_1^3(x_2 - x_1)} + \frac{2.46}{12566x_1^2} - 1.0 \leq 0 \quad \text{stress constraint} \\ g_3(x) &= 1.0 - \frac{140.54x_1}{x_2^2 x_3} \leq 0 \quad \text{surge wave frequency constraint} \\ g_4(x) &= \frac{x_2 + x_1}{1.5} - 1.0 \leq 0 \quad \text{outer diameter constraint} \end{aligned} \qquad (13)$$

with the bounds $0.05 \leq x_1 \leq 0.20$, $0.25 \leq x_2 \leq 1.30$, and $2 \leq x_3 \leq 15$.

Table 3 summarizes and compares the results obtained by applying both the MPS and CiMPS methods (30 runs for each). The number of cheap points generated at each iteration is $N = 100$ and the number of contours used is 5 for all runs. As it can be seen, the CiMPS method results in significantly lower number of constraint checks (ncc) and also less number of function evaluations (nfe).

This problem has been solved by a number of researchers: [12](a numerical technique called constraint correction at constant cost (CCC)), [21] (a GA-based algorithm), [20] (IHS, an improved variation of Harmony Search algorithm [22]), [19] (using eight numerical techniques with the best solution obtained using M-4 method which is a variation of a Lagrange multipliers code based on Powell's algorithm [23]). The above solutions are compared with the best solution found using our proposed CiMPS technique and the original MPS method in Tab. 4. As it can be seen, the solution obtained using our proposed CiMPS method (and the MPS) is better than the ones obtained by other techniques.

4.2.2 Pressure Vessel Design The second problem is to minimize the total cost, including the cost of material, forming and welding of a cylindrical vessel which is capped at both ends by hemispherical heads as shown in Fig. 7.

The total cost $f_{PV}(x)$ is given as

$$f_{PV}(x) = 0.6224x_1x_3x_4 + 1.7781x_2x_3^2 + 3.1661x_1^2x_4 + 19.84x_1^2x_3 \qquad (14)$$

where $x_1 = T_s$ is the thickness of the shell, $x_2 = T_h$ is the thickness of the head, $x_3 = R$ is the inner radius, and $x_4 = L$ is the length of

TABLE 1: Summary information of the test problems and run settings: : N, the number of cheap points generated at each iteration, the number of contours by which the cheap points are grouped, and m the number of expensive sample points generated at each iteration (the quadratic fitting accuracy R^2 is set to 0.999 for all test problems). For a detailed explanation of these settings and their effects one can refer to [1].

					Run settings		
$f(x)$	n	Domain	# constraints	Global minimum	N	m	# contours
QF	2	$[-3.0, 3.0]^2$	2	0.0	100	5	5
SC	2	$[-2.0, 2.0]^2$	4	-1.032	100	5	10
GP	2	$[-2.0, 2.0]^2$	4	3.0	100	5	20
HN	6	$[0.0, 1.0]^6$	2	-3.322	100	30	20
F16	16	$[-1.0, 0.0]^{16}$	4	25.875	500	155	20

TABLE 2: Simulation results obtained for the test problems using both the MPS and CiMPS methods: Minimum denotes the optimal value found; nfe, # function evaluations, ncc, # constraint checks (var, variations, ave: average, med: median)

		Minimum		nfe		ncc	
$f(x)$	Method	var	med	ave	med	ave	med
QF	MPS	[0.000, 0.000]	0.000	17.1	16.5	336.6	309.0
	CiMPS	[0.000, 0.000]	0.000	23.5	23.5	**85.4**	**72.0**
SC	MPS	[-1.032, -0.373]	-1.031	94.2	78.0	3307.7	2694.0
	CiMPS	[-1.032, -1.003]	-1.031	85.0	72.5	**282.2**	**243.5**
GP	MPS	[3.000, 4.043]	3.001	304.7	287.5	4162.9	3836.0
	CiMPS	[3.000, 3.408]	3.001	267.4	231.0	**485.8**	**356.5**
HN	MPS	[-3.319, -2.057]	-3.291	704.5	719.0	24799.5	25398.0
	CiMPS	[-3.321, -2.787]	-3.295	388.1	396.0	**978.0**	**950.0**
F16	MPS	[25.876, 25.902]	25.880	333.2	222.0	4906.8	2413.5
	CiMPS	[25.876, 25.886]	25.878	291.9	245.0	**665.6**	**622.5**

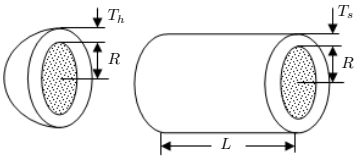

FIGURE 7: A Pressure Vessel

the cylindrical section, subject to the following six constraints

$$
\begin{aligned}
g_1(x) &= -x_1 + 0.0193x_3 \leq 0 \\
g_2(x) &= -x2 + 0.00954x_3 \leq 0 \\
g_3(x) &= -\pi x_3^2 x_4 - \frac{4}{3}\pi x_3^3 + 1296000 \leq 0 \\
g_4(x) &= x_4 - 240 \leq 0 \\
g_5(x) &= 1.1 - x_1 \leq 0 \\
g_6(x) &= 0.6 - x_2 \leq 0
\end{aligned}
\tag{15}
$$

with the bounds $1.0 \leq x_1 \leq 1.375$, $0.625 \leq x_2 \leq 1.0$, $25 \leq x_3 \leq 150$, and $25 \leq x_3 \leq 240$

Table 5 summarizes and compares the results obtained by applying both the MPS and CiMPS methods (30 runs for each). The number of cheap points generated at each iteration is $N = 200$ and the number of contours used is 20 for all runs. As it is seen, the CiMPS method results in significantly lower number of constraint checks (ncc) and also less number of function evaluations (nfe). This problem has been solved by [24] using a GA-based approach, by [25] using Harmony Search (HS) algorithm [22], by [20] using IHS method, an improved variation of the Harmony Search algorithm, and by [26] using branch and bound method. The best solutions obtained using above techniques are compared with the best solution found using our proposed CiMPS technique and the original MPS method in Tab. 6. As it can be seen, the solution obtained using our proposed

TABLE 3: Results obtained for the Minimization of Weight of the Spring problem using both the MPS and CiMPS methods: Minimum denotes the optimal value found; nfe, # function evaluations, *ncc*, # constraint checks (var, variations, ave: average, med: median)

	Minimum		nfe		*ncc*	
Method	var	med	ave	med	ave	med
MPS	[0.012664, 0.013000]	0.01268	32.9	29.0	20994.1	14847.5
CiMPS	[0.012665, 0.013040]	0.01268	21.0	19.5	**1910.6**	**1816.5**

TABLE 4: Best solution obtained for the Minimization of Weight of the Spring problem using the CiMPS method compared with the best solutions reported by other works

	Method					
Design	CiMPS (this work)	MPS [1]	CCC [12]	GA-based [21]	M-4 [19]	IHS [20]
x_1	0.05156	0.05154	0.053396	0.051989	0.0500	0.05115438
x_2	0.35363	0.35322	0.399180	0.363965	0.3176	0.34987116
x_3	11.47221	11.49692	9.185400	10.890522	14.027	12.0764321
$f_{WS}(x)$	**0.012665**	**0.012664**	0.012730	0.012681	0.01272	0.0126706

TABLE 5: Results obtained for the Pressure Vessel Design problem using both the MPS and CiMPS methods: Minimum denotes the optimal value found; nfe, # function evaluations, *ncc*, # constraint checks (var, variations, ave: average, med: median)

	Minimum		nfe		*ncc*	
Method	var	med	ave	med	ave	med
MPS	[7163.73957, 7163.73957]	7163.73957	62.4	61.0	4565.2	4351.0
CiMPS	[7163.73957, 7163.73957]	7163.73957	37.4	37.0	**335.6**	**332.5**

TABLE 6: Best solution obtained for the Pressure Vessel Design problem using the CiMPS method compared with the best solutions reported by other works

	Method					
Design	CiMPS (this work)	MPS [1]	GA-based [24]	HS [25]	IHS [20]	BB [26]
x_1	1.10000	1.10000	1.125	1.125	1.125	1.125
x_2	0.625	0.625	0.625	0.625	0.625	0.625
x_3	56.99482	56.99482	58.1978	58.2789	58.29015	48.97
x_4	51.00125	51.00125	44.2930	43.7549	43.69268	106.72
$f_{PV}(x)$	**7163.73957**	**7163.73957**	7207.494	7198.433	7197.730	7980.894

CiMPS method (and the MPS) is better than the ones obtained by other techniques.

5 Conclusions

Regarding the Mode Pursuing Sampling (MPS) method proposed in [1], it was very important and desirable to handle expensive constraints without significantly changing or badly affecting other MPS's advantages. It means keeping each responsibility (i.e., handling expensive objective functions and constrains) as independent as possible. The CiMPS method proposed in this paper properly ensures this property.

The performance of the CiMPS method was experimentally verified through two test suites, namely, five benchmark functions and two design problems. The CiMPS and its parent algorithm (MPS) was compared on both test suites, also

both competed with four other well-known optimization methods on design problems. The reported results clearly confirmed that, CiMPS outperforms not only the MPS but also other four well-known competitors in terms of convergence speed, number of constraint evaluations, and solution accuracy. By this way, CiMPS opens a promising direction to tackle with expensive constraint optimization problems.

Developing a mixed-type variable CiMPS, benchmarking it on more comprehensive and complex test suites, and enhancing that to solve large-scale problems more efficiently are our directions for the future work.

REFERENCES

[1] Wang, L., Shan, S., and Wang, G., 2004. "Mode-pursuing sampling method for global optimization on expensive black-box functions". *Engineering Optimization,* ***36***(4), pp. 419–438.

[2] Simpson, T., Peplinski, J., Koch, P., and Allen, J., 2001. "Metamodels for computer-based engineering design: survey and recommendations". *Engineering with Computers,* ***17***(2), pp. 129 – 50.

[3] Wang, G., and Shan, S., 2007. "Review of metamodeling techniques in support of engineering design optimization". *ASME Journal of Mechanical Design,* ***129***(4), pp. 370 – 80.

[4] Schonlau, M., Welch, W. J., and Jones, D. R., 1998. "Global versus local search in constrained optimization of computer models". *Lecture Notes-Monograph Series,* ***34***, pp. 11–25.

[5] Regis, R. G., and Shoemaker, C. A., 2005. "Constrained global optimization of expensive black box functions using radial basis functions". *J. of Global Optimization,* ***31***(1), pp. 153–171.

[6] Sasena, M. J., Papalambros, P., and Goovaerts, P., 2002. "Exploration of metamodeling sampling criteria for constrained global optimization". *Engineering Optimization,* ***34***, pp. 263–278.

[7] Yannou, B., Simpson, T. W., and Barton, R., 2005. "Towards a conceptual design explorer using metamodeling approaches and constraint programming". In ASME Design Engineering Technical Conferences - Design Automation Conference, no. DETC2003/DAC-48766.

[8] Yannou, B., Moreno, F., Thevenot, H., and Simpson, T., 2005. "Faster generation of feasible design points". In ASME Design Engineering Technical Conferences - Design Automation Conference, no. DETC2005/DAC-85449.

[9] Moghaddam, R., Wang, G., Yannou, B., and Wu, C., 2006. "Applying constraint programming for design space reduction in metamodeling based optimization". In The 16th International Institution for Production Engineering Research (CIRP) International Design Seminar, no. 10081.

[10] Yannou, B., and Harmel, G., 2006. *Advances in Design*. Springer London, ch. Use of Constraint Programming for Design, pp. 145–157.

[11] Wang, G., 2003. "Adaptive response surface method using inherited latin hypercube design points". *Transactions of the ASME. Journal of Mechanical Design,* ***125***(2), pp. 210 – 20.

[12] Arora, J., 2004. *Introduction to Optimum Design*. Elsevier Academic Press.

[13] Coello, C., 2002. "Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art". *Computer Methods in Applied Mechanics and Engineering,* ***191***(11-12), pp. 1245 – 87.

[14] Mezura-Montes, E., ed., 2009. *Constraint-Handling in Evolutionary Optimization*. Springer.

[15] Azarm, S., and Mourelatos, Z. P., 2006. "Robust and reliability-based design". *Journal of Mechanical Design,* ***128***(4), pp. 829–831.

[16] Sharif, B., Wang, G., and ElMekkawy, T., 2008. "Mode pursuing sampling method for discrete variable optimization on expensive black-box functions". *Journal of Mechanical Design,* ***130***(2), pp. 021402–1–11.

[17] Duan, X., Wang, G., Kang, X., Niu, Q., Naterer, G., and Peng, Q., 2009. "Performance study of mode-pursuing sampling method". *Engineering Optimization,* ***41***(1), pp. 1–21.

[18] Fu, J., and Wang, L., 2002. "A random-discretization based monte carlo sampling method and its applications". *Methodology and Computing in Applied Probability,* ***4***(1), pp. 5 – 25.

[19] Belegundu, A. D., and Arora, J. S., 1985. "A study of mathematical programming methods for structural optimization. Part II: Numerical results". *International Journal for Numerical Methods in Engineering,* ***21***(9), pp. 1601–1623.

[20] Mahdavi, M., Fesanghary, M., and Damangir, E., 2007. "An improved harmony search algorithm for solving optimization problems". *Applied Mathematics and Computation,* ***188***(2), pp. 1567 – 1579.

[21] Coello, C. A. C., and Mezura-Montes, E., 2002. "Constraint-handling in genetic algorithms through the use of dominance-based tournament selection". *Advanced Engineering Informatics*.

[22] Geem, Z., Kim, J., and Loganathan, G., 2001. "A new heuristic optimization algorithm: Harmony search". *Simulation,* ***76***(2), pp. 60–68.

[23] Powell, M., 1978. "Algorithms for nonlinear constraints that use lagrangian functions". *Mathematical Programming,* ***14***(1), pp. 224–248.

[24] Wu, S., and Chow, P., 1995. "Genetic algorithms for nonlinear mixed discrete-integer optimization problems via meta-genetic parameter optimization". *Engineering Optimization,* ***24***(2), p. 137159.

[25] Lee, K., and Geem, Z., 2005. "A new meta-heuristic algorithm for continues engineering optimization: harmony search theory and practice". *Computer Methods in Applied Mechanics and Engineering,* ***194***(36-38), pp. 3902–3933.

[26] Sandgren, E., 1990. "Nonlinear integer and discrete programming in mechanical design optimization". *ASME Jounral of Mechanical Design,* ***112***(2), pp. 223–229.