Proceedings of the
46th IEEE Conference on Decision and Control
New Orleans, LA, USA, Dec. 12-14, 2007

Efficient Derivative-Free Optimization

Paul Belitz and Thomas Bewley

Abstract— **The present paper considers the derivative-free optimization of expensive non-smooth functions. One of the most efficient algorithms for this class of problems is the surrogate-based optimization framework by Booker *et al*, 1999. Searches performed using this algorithm are restricted to points lying on an underlying grid to keep function evaluations far apart until convergence is approached. Once convergence on this discrete grid is obtained, the grid is refined and the process repeated. All previous implementations of this algorithm have been based on a Cartesian grid. However, Cartesian grids are not nearly as uniform at packing, covering, and quantizing parameter space as several alternatives that are well known in coding theory, referred to as "n-dimensional sphere packings" or "lattices". Also, the distribution of nearest-neighbor lattice points turns out to be far superior in these alternative lattices, further increasing the efficiency of the optimization algorithm. The present study illustrates how such lattices may be incorporated into the surrogate-based optimization framework.**

I. Introduction

The minimization of computationally expensive, high dimension functions is most efficiently performed by use of a gradient-based optimization routine. However, when the function in question is too noisy to optimize using gradient information, a derivative-free optimization scheme must be used. An effective class of strategies designed to accomplish this, known as Generalized Pattern Search (GPS) algorithms, coordinates the search for the minimum by performing function evaluations on a discrete grid covering the feasible region of parameter space, refining this grid as convergence is approached.

The most efficient strategies in this vein leverage the use of inexpensive intermediate *surrogate* functions (often, Kriging interpolations are used) to interpolate the available function evaluations in order to provide suggested regions of parameter space in which to perform new function evaluations (see Booker et al., 1999). When exploratory *searches* based on such surrogate functions indicate that a discrete minimizer (a.k.a. *Candidate Minimum Point*, or *CMP*) on the current grid might have been attained, the function is evaluated at several neighboring *poll* points on the grid. To ensure convergence, these points are chosen in such a way as to form a positive basis around the CMP [Torczon 1997, Booker *et al.* 1999, Coope & Price 2001]. That is, any feasible point in the neighborhood of the CMP can be reached via a linear combination *with non-negative coefficients* of the vectors from the CMP to these poll points.

If during this polling step a reduced function evaluation is calculated, the surrogate interpolating function is updated and explored further to identify a new CMP. If, however, all neighboring poll points represent increased function values from the CMP, the CMP is referred to as the best known local minimizer on the discrete grid. The grid is then refined by a factor of two and the process repeated, reusing all previously-calculated function values from the coarser grid. To the best of our knowledge, all previous implementations of such grid-based optimization strategies have been based on Cartesian grids.

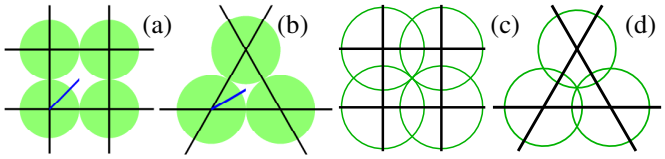

Fig. 1. The packing (a and b) and covering (c and d) of the 2D Cartesian and hexagonal lattices.

II. Lattice Performance Metrics

There are two key drawbacks with such Cartesian grids for the coordination of a derivative-free optimization scheme. First, the Cartesian grid is relatively *nonuniform* compared to the available alternatives. As illustrated in two dimensions in Figure 2, the characteristics of a lattice may be quantified by the following measures [Conway & Sloane 1999]:

- The *packing radius* of a lattice, ρ, is the maximal radius of the spheres in a set of identical nonoverlapping spheres centered at each nodal point.
- The *packing density* of a lattice, Δ, is the fraction of the volume of the feasible domain included within a set of identical non-overlapping spheres of radius ρ centered at each nodal point on the lattice. Lattices that maximize this metric are referred to as *close-packed*.
- The *covering radius* of a lattice, R, is the maximum distance between any point in the feasible domain and the nearest nodal point on the lattice.
- The *covering thickness* of a lattice, Θ, is the number of spheres of radius R containing an arbitrary point in the domain, averaged over the entire domain.
- The *kissing number* of a lattice, τ, is defined as the number of nearest neighbors to any given nodal point in the lattice. In other words, it is the number of spheres of radius ρ centered at the nodal points that touch, or 'kiss', a sphere centered at any given nodal point.
- The *Voronoi cell* of a nodal point on a lattice, $V(P_i)$, consists of all points in the domain that are at least as close to the nodal point P_i as they are to any other nodal point P_j.
- The *mean squared quantization error per dimension* of a lattice, G, is the average mean square distance of any point in the domain and the nearest nodal point, normalized

n	lattice	Δ	Θ	G	τ
2	cubic ($\mathbf{Z}^2$)	0.785	1.571	0.08333	4
	hexagonal (A_2)	0.907†	1.209†	0.08019†	6†
3	cubic ($\mathbf{Z}^3$)	0.524	2.721	0.08333	6
	FCC (D_3)	0.740†	2.094	0.07875	12†
	BCC (D_3^*)	0.680	1.464†	0.07854†	8
4	cubic ($\mathbf{Z}^4$)	0.308	4.934	0.08333	8
	checkerboard (D_4) $\cong$ staggered (D_4^*)	0.617†	2.467	0.07660‡	24†
8	cubic ($\mathbf{Z}^8$)	0.0159	64.94	0.08333	16
	checkerboard (D_8)	0.127	32.47	0.07591	112
	staggered (D_8^*)	0.0317	8.117	0.07474	16
12	cubic ($\mathbf{Z}^{12}$)	3.26e-4	973.4	0.08333	24
	checkerboard (D_{12})	0.0104	486.7	0.07710	264
	staggered (D_{12}^*)	6.52e-4	30.42	0.07480	24

Table 1. Characteristics of some representative lattices: the packing density Δ, the covering thickness Θ, and the mean squared quantization error per dimension, G, quantifying the lattice uniformity, and the kissing number τ indicating the flexibility available in selecting the positive basis from nearest neighbors. The dagger (†) denotes a value *known* to be optimal among all lattices at that n, whereas the double dagger (‡) denotes a value *thought* to be optimal.

by the appropriate power of the volume of the Voronoi cell, and divided by n. Shifting the origin to be at the centroid of a Voronoi cell $V(P_i)$, it is given by $G = \frac{1}{n}\int_{V(P_i)}|x|^2\,dx/\left[\int_{V(P_i)}dx\right]^{1+\frac{2}{n}}$.

Note that the packing density Δ, the covering thickness Θ, and the normalized mean-squared quantization error G are related but distinct quantifications of the uniformity of the lattice, whereas the kissing number τ is an indicator of the degree of flexibility available when selecting a positive basis from nearest neighbors on the lattice.

As depicted in Table 1 for $n = 2,3,4,8$, and 12, the packing density, covering thickness, and mean squared quantization error per dimension are all improved by moving from the Cartesian grid to the available alternatives shown, with this improvement becoming ever more apparent as n is increased. In high dimensions, the Cartesian grid offers poor performance compared to the alternatives. The kissing number of the Cartisian grid is low compared to other lattices - this indicates a lack of flexibility in selecting a poll set from the nearest neighbors, preventing more efficient use of previously evaluated points. Reusing previously evaluated points reduces the number of new function evaluations necessary to complete each poll set, futher improving the GPS algorithm's overall efficiency.

The second key drawback of the Cartesian grid for the coordination of a derivative-free optimization is the poor configuration of the nearest-neighbor grid points for establishing a positive basis around a CMP with a minimal number of additional function evaluations. This is a key step to assure convergence of the GPS algorithm. Since evaluating

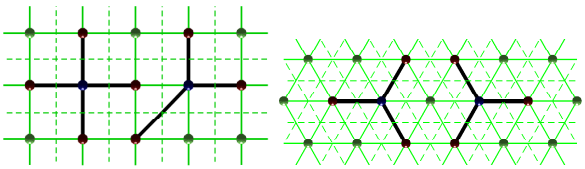

Fig. 2. Layout of the nodes in the 2D Cartesian (left) and hexagonal (right) lattices.

the function is computationally expensive, minimizing the number of points to poll while establishing a positive basis around the CMP is of key importance to maximize the efficiency of the optimization algorithm. Note that a complete poll step performed on a positive basis constructed using nearest neighbors of a CMP in n dimensions requires $2n$ function evaluations on a Cartesian grid, wheareas in more well-behaved lattices such as the Hexagonal lattice, the positive basis requires only $n+1$ points (see Fig 2). In fact, most alternative lattices developed as n-dimensional sphere packings require only $n+1$ points to form a positive basis using nearest neighbor points. Thus, independent from the benefits in increased uniformity provided by the alternative lattices considered here, a factor of almost two increased efficiency can be realized simply due to the more convenient locations of the nearest neighbor points.

A final point worth noting is that it is possible to construct a positive basis with only $n+1$ points, referred to as a *minimal positive basis*, in the n-dimensional Cartesian case if points other than nearest neighbor points are used, as indicated in 2D in Figure 2. However, note that one of the vectors forming the positive basis in this case is $\sqrt{n}$ longer than the other n. In addition, this oddball vector is at a much larger angle to all of the other vectors in the positive basis as these vectors are to themselves. The upshot of both of these facts is that the region to which the optimal point is effectively localized via polling a set of points distributed in such a fashion is greatly increased from that tight region resulting from a poll on a perfectly distributed minimal positive basis on nearest neighbor points. That is, the uniformity of the poll set's radii and angles are both disrupted, skewing the results of the poll. Thus, a Cartesian grid by nature hinders the process of finding a minimal, evenly-distributed positive basis, thereby decreasing the speed of the GPS algorithm.

III. Alternative Lattices - A_n, A_n^*, D_n, and D_n^*

As discussed above, nearly all lattices more complex than Cartesian offer far superior performance as quantified by the metrics outlined above. Four of these lattices include the A_n, A_n^*, D_n, and D_n^*. All but one are defined in $n+1$ dimensions and share very similar structure. All offer excellent performance over a wide range of function dimensions, and significant improvements over Cartesian grids. The similarity in their definitions makes it straightforward to utilize the same algorithms to determine poll sets.

The familiar Cartesian grid is defined by a matrix of orthogonal basis vectors. Any point in the lattice space $\mathbb{R}^n$

can clearly be written as an integer linear combination of basis vectors. For consistency with the published literature on this topic (see Conway and Sloan 1999), we assemble these vectors as the columns of M^T, where the basis matrix M is defined such that

$$M = \begin{pmatrix} 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 \\ . & . & \dots & . & . \\ 0 & 0 & \dots & 0 & 1 \end{pmatrix}.$$

The A_n lattice is defined similarly with a matrix of basis vectors. Note that the n-dimensional lattice is defined in $n+1$. Also note that each basis vector is orthogonal to the vector $\mathbf{v} = \begin{pmatrix} 1 & 1 & \dots & 1 \end{pmatrix}^T$, defining the plane on which the lattice is defined. For simplicity the matrix basis vectors here are not normalized; in use the vectors are normalized before scaling to build the desired lattice.

$$M_{A_n} = \begin{pmatrix} 1 & -1 & 0 & \dots & 0 & 0 \\ 0 & 1 & -1 & \dots & 0 & 0 \\ . & . & . & \dots & . & . \\ 0 & 0 & 0 & \dots & 1 & -1 \end{pmatrix}$$

The A_n^* lattice is defined similarly to A_n, with the final row in the basis matrix being dependent on the dimension. Again, for simplicity, vectors are not scaled.

$$M_{A_n^*} = \begin{pmatrix} -1 & 1 & 0 & \dots & 0 & 0 \\ 1 & 0 & -1 & \dots & 0 & 0 \\ . & . & . & \dots & . & . \\ 1 & 0 & 0 & \dots & -1 & 0 \\ \frac{-n}{n+1} & \frac{-n}{n+1} & \frac{-n}{n+1} & \dots & \frac{-n}{n+1} & \frac{-n}{n+1} \end{pmatrix}$$

The D_n lattice is the n-dimensional analog to the 3-dimensional Face Centered Cubic (FCC) lattice, defined by:

$$M_{D_n} = \begin{pmatrix} -1 & -1 & 0 & \dots & 0 & 0 \\ 1 & -1 & 0 & \dots & 0 & 0 \\ 0 & 1 & -1 & \dots & 0 & 0 \\ . & . & . & \dots & . & . \\ 0 & 0 & 0 & \dots & 1 & -1 \end{pmatrix}$$

The D_n^* lattice gives the common Body Centered Cubic lattice in three dimensions, well known in atomic structures and supermarket orange stacks. The basis matrix is defined as follows:

$$M_{D_n^*} = \begin{pmatrix} 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 \\ . & . & . & . & . \\ 0 & 0 & \dots & 1 & 0 \\ 0.5 & 0.5 & \dots & 0.5 & 0.5 \end{pmatrix},$$

These provide four examples of lattices that offer far better performance than the Cartesian grid in terms of the metrics discussed above, where all are defined similarly and are straightforward to implement numerically. Three of the above lattices - A_n, A_n^* and D_n - are n-dimensional lattices defined in $\mathbb{R}^{n+1}$. An intuitive example is A_2, a triangular $2-D$ lattice that lies on a plane in $\mathbb{R}^3$ (see Figure 3). The problem of having a GPS lattice defined in a higher space is that to use the lattice to find a poll set, either the lattice must be redefined in n, or a transformation must be made to map points in $\mathbb{R}^n$ into the lattice space, and back. The latter approach is more easily implemented than the former, and is discussed below.

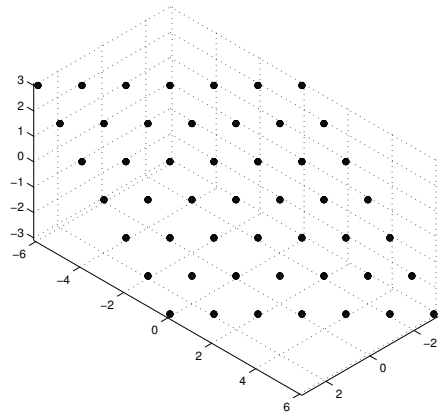

Fig. 3. A_2 as defined on a plane in $\mathbb{R}^3$

The SMF polling algorithm described above consists of four primary steps. First, the CMP is projected onto the plane of the lattice. Second, the closest lattice point to the CMP is identified. Third, the nearest neighbor points surrounding the lattice CMP are found. Fourth, the lattice CMP and neighbors are projected back into parameter space. Finally, a maximally efficient positive basis is constructed from these neighbors.

In the case of A_n, the lattice plane is easily identified by virtue of the lattice basis matrix's orthogonality to the vector $\mathbf{v} = \begin{pmatrix} 1 & 1 & \dots & 1 \end{pmatrix}^T$. The Cartesian plane orthogonal to $\mathbf{v}$ is determined by using the QR Householder algorithm to find the Cartesian basis vectors orthogonal to $\mathbf{v}$. Once the basis matrix M_{plane} of the Cartesian plane is determined, the corresponding point $\mathbf{x_p}$ on the lattice plane to the CMP $\mathbf{x} \in \mathbb{R}^n$ is given by

$$\mathbf{x_p} = M_{plane}^T \mathbf{x}$$

and the reverse transformation is

$$\mathbf{x} = M^T \backslash \mathbf{x_p}$$

using Matlab notation for the linear system solution. This gives a transformation from Cartesian n-space to the Cartesian plane in $n+1$ that the A_n lattice lies on. As the lattice lies on this plane, the reverse transformation does not affect the uniformity of the neighbor set in any way - the neighbor and CMP points do not move in relation to one another when the transformation is performed. Thus, the uniformity of both angles and distances is perfectly perserved.

Once a CMP in parameter space has been mapped onto the A_n plane, the nearest lattice point must be identified. The plane that A_n is defined on is the set $P \in \mathbb{R}^{n+1}$. The CMP is $\mathbf{x} \in \mathbb{R}^n$. First, $\mathbf{x}$ is mapped into P, defining the CMP on the lattice: $\mathbf{x_p} = P^T \mathbf{x}$. Next, $\mathbf{x_p}$ is defined in terms of the A_n basis vectors: $\mathbf{x_{A_n}} = M_{A_n} \backslash \mathbf{x_p}$. Now the CMP is defined in $\mathbb{R}^{n+1}$ as a linear combination of the lattice basis vectors. That is, the CMP is redefined as a linear combination of the lattice basis

vectors. To find the nearest lattice point, x_{A_n} is scaled to the appropriate lattice scale, and rounded to the nearest integer. This gives the CMP on the lattice in terms of the lattice basis vectors.

The nearest neighbors surrounding the CMP are defined as all points on the lattice that are a distance r from the CMP, where r is the radius defined by the lattice scale. Thus, to find the neighboring points to the CMP, all possible linear combinations of the basis vectors with the coefficients of -1, 0, and 1 are calculated. All points a distance r from the CMP are kept, given exactly the nearest neighbors of the CMP, defined in $\mathbb{R}^{n+1}$.

Next, the reverse transformation back into $\mathbb{R}^n$ described above is performed, giving the lattice approximation to the CMP, and the nearest neighbors to that point. As many lattices of interest share very similar definitions, this algorithm to find the CMP and nearest neighbors is independent of the lattice being used. Unlike other GPS codes, only the definition of the basis matrix and the corresponding orthogonal vector must be altered to implement a different lattice. This flexibility allows the user to vary the lattice used with minimal algorithm modification.

IV. The Challenge of Finding a Positive Basis

Once the neighbors of the CMP have been identified, a positive basis must be identified as the poll points, insuring convergence of the SMF algorithm.

In $\mathbb{R}^2$ the neighbors of the CMP lie on the unit (normalized) circle around the CMP. In $\mathbb{R}^3$ the neighbors lie on the unit sphere. In higher dimensions the neighbors lie on the dimensionally appropriate hypersphere. The challenge is to construct a positive basis from these neighbors while minimizing the number of points forming the basis without the use of a prohibitively expensive algorithm.

Take a continuous hypersphere in n dimensions. Take $n+1$ points, and let the points behave as charged particles of unit strength - that is, each particle exerts a force on the other particles proportional to the inverse square of the distance. On a continuous hypersphere, $n+1$ points will reach equilibrium forming a minimal positive basis.

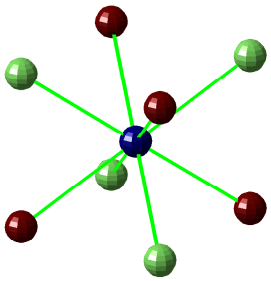

Fig. 4. Two different positive basis on A_3, shown in green and red around the blue CMP. Note the complete radial and angular uniformity as well as the flexibility in the orientation of the basis.

The sphere in the idealized problem described above is discretized by the CMP's nearest neighbors, which by definition lie on a hypersphere. To minimize the number of function evaluations, any previous function evaluations included in the nearest neighbors are by construction preserved in the basis set. The number of such previous evaluations is denoted as l. To find a positive basis, $n+1-l$ additional points are randomly distributed on the locations of the nearest neighbors. Each point is treated as a charged particle of unit strength, as are the l fixed previous function evaluations. At equilibrium on a continuous hypersphere, the greatest force experienced by any point will be minimized. Thus, on the discrete hypersphere, the aim is to minimize the greatest individual force over all $n+1-l$ points. This would most effectively be performed via an exhaustive search, checking for a positive basis when the force has been minimized. If the force is minimized without producing a positive basis, no positive basis can be found with $n+1-l$ points. Another point would have to be added ($n-l+2, 3, 4, \ldots$. total), and the algorithm repeated until a positive basis is found. However, when operating in only even moderately high dimension, the computational cost of this failsafe technique would be prohibitively expensive due to the high number of neighbors. Therefore, a less expensive and less straightforward algorithm is derived using the same force-based minimization concept. This algorithm is described below.

Again, $n+1-l$ points are randomly distributed over the CMP's nearest neighbors. The l fixed points contribute to the force on the points. The total force on each point is calculated. The two points experiencing the greatest force are selected, and moved to unoccupied neighbor locations. The forces on all points are again calculated. If the greatest force is greater than in the initial configuration, the two particles are moved again. If the greatest force is less than in the initial configuration, then the particle set is closer to equilibrium, and the set is tested for a positive basis. If no positive basis has been found, the forces on all particles are recalculated, the (new) two points experiencing the greatest forces are selected and another iteration is performed.

Moving only two points at once (as opposed to iterating all $n+1-l$) is critical to reducing the computational cost of the algorithm; however, the greatest force on a point is often minimized on the discrete set of neighbors without the set of points forming a positive basis. That is, unlike on a continuous hypersphere, the algorithm does not reliably converge to a positive basis for all initial conditions, as the neighbors are a very rough discretization of a hypersphere. The end result of the algorithm is heavily dependent on the random initial conditions of the point set. Tests on the A_n matrix, which has several possible positive basis' with $n+1$ points demonstrate that the two-point iteration algorithm does not always converge to a positive basis for a given initial point distribution.

Therefore, if all combinations of moving two points have been tested and no positive basis has been found, the algorithm is repeated using a new random initialization. Tests on A_n have showed that this repeated random initialization scheme results in a positive basis within a few initializations, demonstrating the feasibility of the algorithm.

Frequently no positive basis exists using only $n+1-l$

points, particularly given poorly configured previous function evaluations. Therefore, the number of new random initializations is limited at a user-set value, beyond which the algorithm is repeated using $n+2,3,4...$ points, until a positive basis is found. While this will occasionally result in an additional poll point or two compared to the optimal solution, the massive increase in the computational speed of the positive basis-seeking algorithm makes for a reasonable tradeoff. Additionally, as the cap on initializations is user-defined, the poll point algorithm can be tuned by the user to best match the given function expense versus speed of the GPS algorithm. As the cost of a function evaluation increases, the user can increase the number of iterations permitted to avoid extraneous poll points at the cost of the GPS code requiring more computational power.

The algorithm descriped above is based on the assumption that the existence of a positive basis can be detected. By definition, a positive basis is a set of vectors that span the space they are defined in - any point in the space can be written as a positive linear combination of the vectors. An equivalent statement is that if all $2n$ Cartesian basis vectors can be written as a positive linear combination of the vectors, then the vectors form a positive basis. When testing for a positive basis, the Matlab linear program function `linprog` is used. The function minimization of `linprog` is of no interest; however the equality constraint $A_{eq}\mathbf{x} = b_{eq}$ is exactly what is needed, where A_{eq} is the matrix of basis vectors that are being tested, b_{eq} is the Cartesian vector being tested, and $\mathbf{x}$ being the vector being solved for. The function to be minimized is set as a null matrix, ensuring that `linprog` does nothing but check whether the equality constraint has a solution. If the equality constraint can be satisfied for all Cartesian basis vectors, where the matrix A_{eq} is the poll vectors, then the poll set forms a positive basis. Thus, `linprog` is called for each Cartesian basis vector. If for every vector the constraint can be satisfied, the poll set is a positive basis.

Tests on A_n have demonstrated the efficacy of this basis-finding algorithm. The algorithm reliably finds a positive basis even when given poorly spaced previous function evaluations, while maintaining a reasonble computational cost. An additional benefit of this algorithm is the lack of any dependence on the lattice being used. The only inputs are the dimension of the problem and the locations of the nearest neighbor points. That the lattice in question plays no role allows for much greater flexibility in the program when applying various lattices in the GPS algorithm.

V. Boundary Conditions

For simplicity we assume rectangular constraints on the parameter space - the limits of allowed function variable values. There are two possible ways the GPS algorithm can violate the boundary constraints; either a poll point lies slightly outside the allow variable bounds, or the CMP is very close to or past the boundary, necessitating a modified polling algorithm.

In the first scenario, the CMP is relatively far from the boundary. What defines 'far' is user-defined, but is generally somewhere between $0.5r$ to r from the boundary, where r is the lattice scale. When the CMP nearest neighbors are checked for boundary compliance, and a small number of points are found to violate the boundary condition by a small amount, a reasonable solution is to move the offending neighbors until each point lies on the violated boundary. This is performed by determining the vector from the CMP to the offending point. Next, the distance along this vector to the violated bound is calculated. This gives the new location of the offending point, preserving the angular uniformity of the neighbor set, and only slightly modifying the radial uniformity of the set. Thus, a positive basis can be found without any greater cost (no modification of the poll point algorithm) than if the bound were less restrictive, with only a small perturbation to the region spanned by the poll set.

When the CMP moves close to or onto the boundary, the above strategy is no longer effective, as the radius of the offending neighbors becomes so small that the parameter space covered by the positive basis does not begin to approximate the desired area. In this case, the function has moved onto the boundary, and in an intuitive sense, the poll points will be restricted to the 'wall' of the boundary, using one additional point to allow the function to move 'off' the 'wall'. In three dimensions, with one violated boundary, this can be visualized with relatively effort. The function space is a cube, and the CMP lies on one wall. The poll points form a positive basis on the wall in two dimensions, and an additional point is chosen that defines a vector perpendicular to the wall. Thus, the wall is explored via the lattice, and the CMP can still move off the wall, back into the allowed space.

Given a parameter space in n dimensions, where the CMP is located on l boundaries, the lattice defining the majority of the poll points is A_{n-l}. So first, the points in unconstrained dimensions are found - the algorithm described above is applied to the A_{n-l} lattice. This poll set will allow movement in the 'open' area of the space, where boundaries are not an issue.

Next, the violated boundaries are considered. For each of the l boundaries, the goal is to find a vector that allows the CMP to move off one boundary at a time. That is, from the CMP, one vector should move back into the allowed function space, staying on $l-1$ boundaries. This process is straightfoward if which particular vector is of no interest. However, the goal is to always remain on a lattice. That is, from l constraints, the algorithm should be able to move along one 'wall' leaving $l-1$ constraints, and then be on a $n-l+1$ lattice.

This procedure is most easily illustrated with an example. If the CMP lies at the point $(3,3,5,5,3)$, where the maximum allowed value is 5, then the third and fourth dimensions are maximally constrained. The vectors moving off one 'wall' at a time are $\mathbf{v_1} = (0,0,-1,0,0)$ and $\mathbf{v_2} = (0,0,0,-1,0)$. However, these two vectors are not guaranteed to lie on a lattice. Thus, a lattice in $n-l+1$ is defined - the lattice that quantizes the boundary 'wall'. From there, the closest point

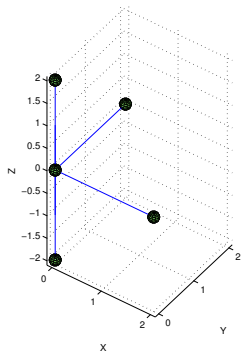

Fig. 5. CMP at (0 0 0), where $X = 0$ and $Y = 0$ are feasibility boundaries. Note the positive A_n basis in Z and the two vectors allowing the algorithm to move off the boundaries, one at a time.

to the CMP and the corresponding nearest neighbors on the $n-l+1$ lattice are found. The vector that is most nearly parallel to $\mathbf{v_1}$ is selected and included in the poll set. The procedure is repeated for $\mathbf{v_2}$, see Figure 5.

This boundary handling algorithm allows the CMP to move along boundaries for long periods without incurring a cost in efficiency, as the CMP is always on a lattice of the appropriate dimension. With the exception of the vectors allowing the CMP to move back into the function space, the majority of the poll points are defined on the same lattice. This algorithm also provides the ability to handle an arbitarily large number of constrained boundaries. Thus, even a very poorly posed problem that runs into multiple boundaries will not prematurely terminate the GPS algorithm. The minimum within the allowed parameter space will be found efficiently regardless of the boundaries.

VI. Conclusion

The Generalized Pattern Search algorithms for performing derivative-free high-dimensional function minimization are well known and effective. However, all codes have thus far used a Cartesian grid to discretize the function space and determine the location of function evaluations. Cartesian grids are known to offer poor performance at quantizing, covering, and packing. Further, in the GPS algorithm, the poll sets produced by Cartesian grids offer far less uniformity both radially and angularly compared to n-dimensional sphere packings, or *lattices*. These alternative lattices offer vast improvements in space quantization, particularly as the dimension of the function increases.

Many lattices exhibit performance characteristics that are desirable in the GPS algorithm. The A_n lattice was selected as a starting point in this work, as it is representative of a class of lattices defined in similar manner, and offers extensive improvements over Cartesian grids. The n-dimensional lattice is defined in $\mathbb{R}^{n+1}$, necessitating the use of a transformation to map points in the function's parameter space $\mathbb{R}^n$ onto the plane where the lattice is defined. Once the neighboring points on the lattice are found, the reverse mapping is necessary to find the appropriate points that the function can evaluate.

The algorithm developed in this work maps an initial point onto the plane of the lattice. The nearest lattice point (the CMP) is found. The neighbors on the lattice are then calculated, and the CMP and neighbors are mapped back into parameter space. There, a positive basis can be found on the neighbors, defining the locations of function evaluations, completing the GPS algorithm.

A unique force-based positive basis search algorithm has been developed that avoids massive computational cost penalties that less sophisticated strategies incurr, without making the algorithm lattice-dependent. The flexibility of the code developed allows for very simple modification to use different lattices in the GPS algorithm. This allows both for the testing of lattice performance in the GPS algorithm application, as well as selecting the lattice to be used based on the character of the function to be minimized. A working code implementing such lattice-based optimization, `Checkers`, is very nearly complete, and our initial comparison of `Checkers` to standard Cartesian-based algorithms will be reported shortly.

VII. Acknowledgments

The authors would like to thank Alison Marsden and Sebastien Michelin for their fruitful collaboration in lattice-based optimization schemes.

References

[1] Booker, A., Dennis, J., Jr., Frank, P., Serafini, D., Torczon, V., Trosset, M., (1999) A rigorous framework for optimization of expensive functions by surrogates, *Structural and Multidisciplinary Optimization*, **17:** 113.

[2] Bewley, T.R., (2007) Numerical Renaissance: Simulation, Optimization, and Control

[3] Marsden, A.L., Wang, M., Dennis, J.E. Jr., and Moin, P., (2004) Optimal aeroacoustic shape design using the surrogate management framework, *Optimization and Engineering*, **5(2):** 235-262. Special Issue on Surrogate Optimization.

[4] Conway, J.H., Sloane, N.J.A, (1999) Sphere Packings, Lattices, and Groups, Third Edition

[5] Charles, A., Dennis, J. JR., Mesh adaptive direct search algorithms for constrained optimization. Submitted to SIAM J. Optim.

[6] Coope, I.D., Price, C.J., (2001) On the convergence of grid-based methods for unconstrained optimization, *SIAM J. Optim.*,**11**:859869

[7] Torczon, V, (1997) On the convergence of pattern search algorithms, *SIAM J. Optim.*, **7**: 1-25.

[8] Jones, D, (2001) A Taxonomy of Global Optimization Methods Based on Response Surfaces, *Journal of Global Optimization.*, **21:** 345-383.

[9] Conn, A.R., Scheinberg, K., Toint, Ph.L., (1998) A Derivative Free Optimization Algorithm in Practice, *American Institute of Aeronautics and Astronautics Conference, Sep 98*