Finding Interesting Things: Population-based Adaptive Parameter Sweeping

Sean Luke
sean@cs.gmu.edu

Deepankar Sharma
dsharma2@cs.gmu.edu

Gabriel Catalin Balan
gbalan@cs.gmu.edu

Department of Computer Science
George Mason University
4400 University Dr, MSN 4A5
Fairfax, VA 22030 USA

ABSTRACT

Model- and simulation-designers are often interested not in the optimum output of their system, but in understanding how the output is sensitive to different parameters. This can require an inefficient sweep of a multidimensional parameter space, with many samples tested in regions of the space where the output is essentially all the same, or a sparse sweep which misses crucial "interesting" regions where the output is strongly sensitive. In this paper we introduce a novel population-oriented approach to adaptive parameter sweeping which focuses its samples on these sensitive areas. The method is easy to implement and model-free, and does not require any previous knowledge about the space. In a weakened form the method can operate in non-metric spaces such as the space of genetic program trees. We demonstrate the method on three test problems, showing that it identifies regions of the space where the slope of the output is highest, and concentrates samples on those regions.

Categories and Subject Descriptors

G.1.6 [**Optimization**]: Gradient Methods

General Terms

Experimentation

Keywords

Parameter Sweeps, Adaptive Sampling, Bracketing

1. INTRODUCTION

Most population-based stochastic search methods try to discover, not surprisingly, *optima* in a search space. But sometimes that's not what an experimenter wants. For example, model-builders in the sciences are often interested less in the optima over their models' parameter spaces than they are in understanding what those spaces *look like*, so as to gain insight into the preexisting natural phenomena from which they have derived their models. Some such models — "swarm"-style multi-agent systems models for example — often have complex dynamics not describable with closed-form mathematical functions, and are poorly understood by the model designer in the first place.

To examine such a model, the experimenter often performs some kind of sweep over its parameter space, sampling points in the space and computing the model behavior at those points. This gives the model-builder data from which he can answer questions such as: what nonlinear relationships exist among the variables; to which variables is the model particularly sensitive; where do rapid changes in model behavior occur; and how might dimensionality be reduced? These parameter sweeps can be expensive depending on model complexity and run-time.

In this paper we propose a population-oriented selection method for performing adaptive parameter sweeps. The method focuses the majority of its time on areas of the space where the model behavior changes, and only sparsely samples those large areas where nothing unusual happens. In this paper we will simplify model behavior to just a single output variable, and so the method translates roughly to an optimization algorithm which finds and emphasizes the sampling of points in those regions of space where the absolute value of the slope of the output — that is, the magnitude of the output gradient — is highest.

If the output function was known and differentiable, we could do this simply by taking the first derivative of the output function and looking for large positive or negative areas. We could then pick samples under some probability distribution heavily weighted towards areas of steepest slope. For example, if the output function were $f(\vec{x})$, we might compute the magnitude of its gradient $g(\vec{x}) = ||\nabla f(\vec{x})||$, and then select new points with a probability proportional to the value of g at each point. This approach has several difficulties. First, it presumes that the function is differentiable. Second, it presumes that we know the function already: but the function is likely derived from the vagaries of the model, and thus we are unlikely to know much about it — hence the need for parameter sweeping!

Another approach is to generate a number of samples, and then fit a curve to the model's collective output values for

GECCO'07, July 7–11, 2007, London, England, United Kingdom.

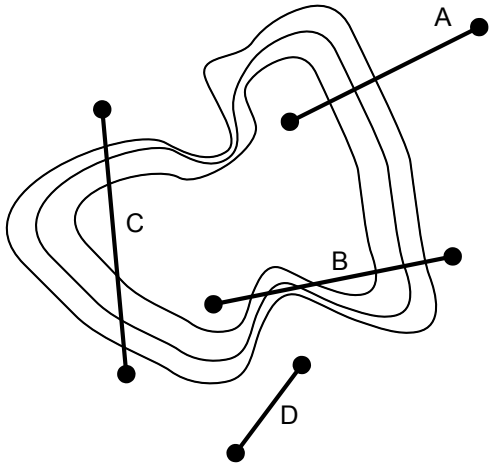


Figure 1: Four different "bracketing lines" traversing a hilly region in the search space.

those samples (known in statistics as the model's *response surface*). We might develop this curve using a neural network, a regression technique, or a mixture of gaussians, for example. Assuming the curve was differentiable, we could then select new points under the magnitude of its gradient similar to the sampling method from the previous paragraph. Unfortunately, constructing this curve requires us to make fairly strong assumptions about the model in order to pick a response surface technique with the appropriate learning bias.

We propose instead a novel approach which performs this adaptive sampling without the need to fit a curve to the model. Instead, we iteratively pick pairs of samples from a preexisting sample set such that the samples' model outputs are very different from one another, and secondarily, such that the two samples are fairly close to one another. We then generate a new sample along the line between the two, using the heuristic that there is very likely a strong slope transition somewhere in-between them. We then add this new sample to the set. We may augment this with a local optimization procedure, repeating the sample-generation along this line some N times in a bracketing fashion, each time using the child to replace the parent closest and most similar in fitness to the child.

The method is population-oriented and bears important relationships with evolutionary computation (EC), so we describe it roughly in EC terms. An evaluated sample in the search space is an *individual* and the collective samples produced so far may be viewed as a *population*. The output of the system at a given sample is equivalent to the *fitness* of an individual, and the procedure we will use to generate new individuals applies certain kinds of *tournament selection* and *crossover* to the population. The algorithm is roughly a steady-state procedure, except that no individuals are ever deleted from the population: it just continues to grow. We will use EC terminology in the remainder of the paper.

2. THE ALGORITHM

Let us assume, for the time being, that our search space is real-valued and multidimensional. Our algorithm repeatedly selects pairs of individuals from the population, crosses them over in a certain fashion to produce a child, and then adds the child to the population. The objective is to produce children which are closer to steep-slope transitions in the search space. The search heuristic is very simple: if two individuals in the population have wildly different fitnesses, then some kind of fitness transition exists in the region between them. Figure 1, line A, shows this situation. Line B shows a related situation where multiple transitions may appear between the two individuals. In either case, at least one transition exists somewhere between the two points. By contrast, if the individuals' fitness is similar to one another, then either there is no transition between them (Figure 1, line D) or there exists an entire hill or valley between them (line C). We have no evidence if the hill or valley exists, and so will ignore this possibility except to include some random exploration to allow for its discovery. Our secondary heuristic is also simple: the closer the individuals are to one another, the more likely that this transition is steep in slope.

Parents are selected as follows. We select the first parent at random from the existing population. We then use a double tournament selection procedure to select the second parent. Specifically, we perform several tournament-selection tournaments, preferring individuals near to the first parent. The winners of these tournaments then compete together in a final tournament preferring the individual which is most different from the first parent in fitness. The winner of this final tournament becomes the second parent.

Once we have selected parents, we then produce a child lying somewhere on the line segment between them. The child is then added to the population. We may then perform a local optimization procedure in the form of iterated bracketing to focus more closely on the steep transitions: given the parents p_1 and p_2 and child c, we replace with c the parent p_i whose difference in fitness with c, divided by the distance between them, is highest. Along the line segment between the revised parents p_j and $p_i = c$ we produce yet another child, add c to the population, and repeat the process.

Iterated bracketing is highly exploitative, and our crossover procedure cannot create children outside the convex hull of the current population. To give some exploration to the procedure we add random individuals to the population in two ways. First, instead of selecting the first parent from the population, occasionally we generate a parent at random from the space, evaluate it, insert it into the population, and select it. Second, we seed the initial population with randomly-generated and evaluated individuals.

The algorithm used is described in pseudocode below. It requires the user to provide several items:

- A CROSSOVER procedure, ideally one which produces a child along the line between two individuals.
- A procedure to CREATE a random individual.
- A procedure to ASSESS the fitness of an individual.
- A procedure DIST to compute the metric distance between two individuals.
- The value *initializationSize*, specifying the initial number of randomly-generated individuals to seed the population.
- The value *exploreProbability*, specifying the likelihood that the first parent will be generated at random rather than chosen from the population.

- The value *numBrackets*, specifying the number of iterations bracketing crossover is performed.
- The values *fitTourn*, and *distTourn*, giving the respective tournament sizes for the fitness-difference tournament and metric-distance tournament.

procedure GRADIENTMAGNITUDESEARCH(
initializationSize, exploreProbability, numBrackets, fitTourn, distTourn)
▷ Population Initialization
$pop = \emptyset$
for $i = 1...initializationSize$ **do**
$ind =$ CREATE a random individual
ASSESS(ind)
$pop = pop \cup \{ind\}$

loop for some time
▷ Parent Selection
with probablity *exploreProbability*
$parent1 =$ CREATE a random individual
ASSESS($parent1$)
$pop = pop \cup \{parent1\}$
otherwise
$parent1 =$ a random individual from pop
$parent2 =$ SELECT($parent1, pop, fitTourn, distTourn$)
▷ Iterated Bracketing
for $i = 1...numBrackets$ **do**
$child =$ CROSSOVER($parent1, parent2$)
ASSESS($child$)
$pop = pop \cup \{child\}$
$parent1 =$ UNLIKEPARENT($parent1, parent2, child$)
$parent2 = child$
return pop

procedure UNLIKEPARENT($parent1$, $parent2$, $child$)
if $\frac{|\text{Fitness}(parent1) - \text{Fitness}(child)|}{\text{DIST}(parent1, child)} > \frac{|\text{Fitness}(parent2) - \text{Fitness}(child)|}{\text{DIST}(parent2, child)}$ **then**
return $parent1$
else
return $parent2$

procedure SELECT($parent1$, pop, $fitTourn$, $distTourn$)
$best =$ SELECT2($parent1, pop, distTourn$)
for $i = 2...fitTourn$ **do**
$new =$ SELECT2($parent1, pop, distTourn$)
if $|\text{Fitness}(new) - \text{Fitness}(parent1)| > |\text{Fitness}(best) - \text{Fitness}(parent1)|$ **then**
$best = new$
return $best$

procedure SELECT2($parent1$, pop, $distTourn$)
$best =$ a random individual from $pop - \{parent1\}$
for $i = 2...distTourn$ **do**
$new =$ a random individual from $pop - \{parent1\}$
if $\text{DIST}(new, parent1) < \text{DIST}(best, parent1)$ **then**
$best = new$
return $best$

Notes. First, care must be taken to not allow crossover to be a "traditional" crossover which picks genes from one parent or the other. Such a crossover generates children at a corner of the hypercube bounding the two parents, and not along the line "between" them. Hypercube corners are less likely to lie on the slope transition between the parents.

Second, we believe that this algorithm can be used in a *non-metric* space, such as the space of genetic program trees. To do this, we need a crossover procedure which produces children which are arguably a "blending" or "averaging" of their parents. In such a space the DIST procedure is unavailable: we simply rely on the (weaker) fitness-difference-only heuristic, setting $\forall x, y$ $\text{DIST}(x, y) = 1$. Of course, if even an approximate distance metric was available, it might be used in the stronger heuristic form, albeit less reliably.

Third, notice that the population is static and continues to grow as new children are added to it. It is important to understand why. We are *not* performing just an optimization procedure — we are doing a sweep of the parameter space. The end result of this sweep is a set of points to assist the experimenter in understanding this space. Thus the only reason we'd ever want to *remove* individuals from the population is if they are hindering this goal. Some possible justifications for removing points include: bounds on physical memory; and a dense cluster of individuals which constantly generates new, useless children within that cluster.

3. RELATED METHODS

Bracketing. Bracketing methods have of course long been used for other purposes. For example, *bisection* finds the zeros of a function f by first selecting two points i and j on either side of the zero, such that $f(i)$ and $f(j)$ have opposite signs. Then the following replacement procedure is iterated: k is set to $\frac{i+j}{2}$, the point mid-way between i and j. If $f(k)$ has the same sign as $f(i)$, i is set to k, else j is set to k. This eventually moves i and j closer to the zero point until they are within some tolerance ϵ. A related but faster-converging method, *regula falsi*, selects a more accurate middle point k based on the relative distances $f(i)$ and $f(j)$ from zero. Here, $k = \frac{if(j) - jf(i)}{f(j) - f(i)}$.

Bracketing can also be used to perform optimization. The *Brent-Dekker* method, sometimes known as *Brent's method*, finds the minimum of f by first choosing three arbitrary values $x < y < z$, where x and z are on opposite sides of the minimum. It then iterates, extracting the quadratic curve which fits the points $\langle x, f(x)\rangle$, $\langle y, f(y)\rangle$, and $\langle z, f(z)\rangle$, and computing the minimum a of that curve. If $x < a < y$, then z is set to y and y is set to a. Else if $y < a < z$, then x is set to y and y is set to a. Eventually the three values x, y, and z converge to the minimum of f.

These methods are primarily intended for searching in one-dimensional spaces, and to look for zero crossings or optima rather than steep slopes. But it is interesting to note that versions of them could be modified to operate in a framework similar to the algorithm described here. For example, we might search for zero crossings in the space by picking two parents which have opposite signs in the fitness function (and ideally *close* in fitness), rather than ones which are merely very different in fitness.

Sampling and Experimental Design. Experimental design [3, 9] is a well-trodden area of statistics, and sampling is its central issue. The key assumption in this field is that experiments are extremely time consuming, and so one should try to minimize the number of samples without overlooking important features of the parameter space.

Many experimental sampling layouts have been proposed, both deterministic and stochastic. The most basic layout is *factorial design*, where all combinations of parameter values are explored; this layout is suitable for understanding the interaction among parameters, but can only be applied to situations where the domains are discrete (or can be reasonably discretized) and the dimensionality is low. The large number of required samples can be decreased by using *fractional factorial design*, where only a constant fraction (typically $\frac{1}{2}$ to $\frac{1}{4}$) of the combinations are sampled. Another commonly used sampling layout is *central composite design*, which surrounds each of the (fractional) design sample points with additional points, so the curvature of the response surface can be estimated. Common stochastic methods include random sampling and the *latin hypercube* [13, 8], where randomly-generated samples are accepted as long as they do not lie on the same (discretized) row, column, etc. as any previous sample point.

The emphasis in such sampling is to sample the entire space approximately uniformly, rather than finding unusual regions and concentrating on them: indeed, sparse semi-uniform sampling may entirely miss steep transitions in the space which lie "between the cracks" so to speak. In contrast, our method will identify those transitions.

Fitting Curves to Response Surfaces. The classic experimental design literature produces approximations of the response surface using polynomials of degree one or two [14, 13], selecting polynomial coefficients so as to minimize the sum of squared errors in the sample points. Optimized higher degree polynomials have been attempted using genetic programming and local (derivative-based) smoothing [16]. An extension [15] augmented the work with an approximate model with partial interpolation.

Estimation of Distribution Algorithms. There are other ways of selecting points under distributions besides fitting a curve to the existing population. Consider that a point in space is a tuple of values, one per parameter. We might generate a new point by picking values for each parameter using a per-parameter distribution based on the fitness of past individuals with various values for that parameter. This treats all the parameters as independent of one another, and is the basis for a number of Estimation Distribution Algorithms (EDAs) such as the Univariate Marginal Distribution Algorithm [10], Population Based Incremental Learning [2], and the Compact Genetic Algorithm [4]. Extensions may consider linkages among the parameters: for example, the Bayesian Optimization Algorithm [11] models such relationships sparsely using a bayesian network.

We mention EDAs because they are the natural example in the evolutionary computation world of methods which generate children by sampling under fitness distributions of individuals in the population, although generally not under curves fit to response surfaces. One connection to curve fitting lies in the use of gaussians to represent individual parameters for continuous-domain problems. Examples include Stochastic Hill-Climbing with Learning by Vectors of Normal Distribution [12], the continous-domain version of Mutual Information Maximization for Input Clustering [6], and the Estimation of Multivariate Normal Algorithm [7]. Such curves are only marginal or bivariate, and these EDAs are, of course, meant for optimization and not slope-finding. Nonetheless, adaptations taken from EDAs might prove fruitful in future versions of this algorithm or in comparison to it.

Probabilistic Roadmap Techniques. A common problem in robotics is finding a route, through the configuration space of the robot, from its initial configuration to some goal configuration: for example, charting a path for a mobile robot to a goal location while avoiding obstacles. The popular *probabilistic road map* (PRM) methods randomly sample this configuration space, reject samples from invalid configuration regions (a robot inside an obstacle, say), and build a traversal graph from among the remaining samples.

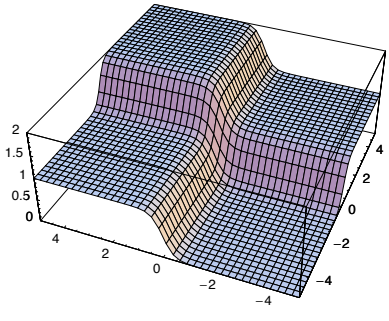
One difficulty with PRM methods is that uniform sampling is unlikely to find those crucial points in the narrow passages of the configuration space, and so it is desirable to sample more heavily near obstacle boundaries or other regions likely to be those narrow passages. This leads to bracketing approaches related to our technique. One method [1] identifies points close to boundaries by choosing an invalid configuration point, then a valid one, and iteratively choosing new points along the line segment between them via bracketed bisection. Another technique, known as the *bridge method* [5], first selects two invalid regions which tend to be near one another; then if the midpoint on the line segment between them lies in a valid region, it is selected. Thus the line segment "bridges" a narrow passage.

These methods use forms of bracketing to favor boundary regions over large open spaces, but they differ from our technique in that they exist in spaces where every point is of one of two values ("valid" or "invalid"), and so the "slope" between them is infinite. Thus they do not consider the degree of difference between two points; only that they *are* different.

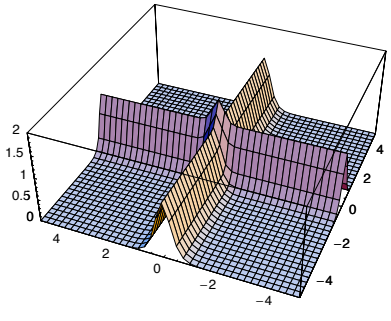
4. EXPERIMENTS

Because the technique is new and its problem domain is relatively novel, the initial experiments in this paper are aimed towards visual and empirical verification of the technique, with an emphasis on problem spaces with large "uninteresting" areas separated by "interesting" border transitions. We hope to perform a more in-depth analysis in future work.

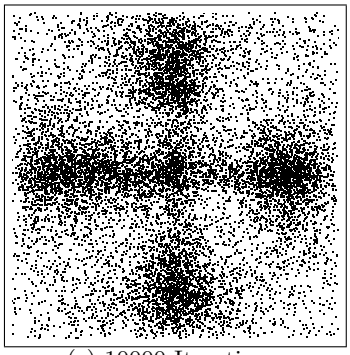
We performed the same basic experiment fifty times over three different test-cases and collected the mean results over the fifty runs. For each run, we iterated the method 10000 times, then extracted data to compare the density of search points on patches of the problem landscape, ordered by the slope of the landscape. To perform this extraction, we computed the slope (the magnitude of the gradient) at each search point generated by the method. We then discretized this slope in units of 1/100, and added it to a bucket for that slope discretization. We then performed one million uniform random samples over the environment, and placed them into discretized slope buckets as well. Then, for each discretization level, we divided the number of run samples at that discretized slope by the number of uniform random



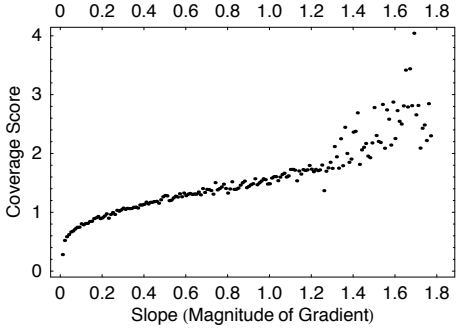
(a) Function $Cross(x, y)$



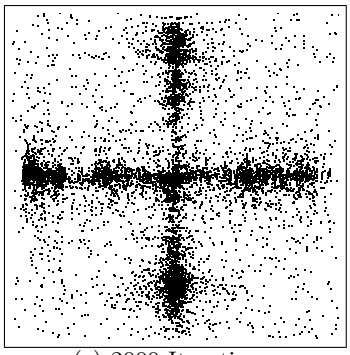
(b) Slope: $||\nabla Cross(x, y)||$



(c) 10000 Iterations, $numBrackets = 1$



(d) Coverage Score by Slope, 10000 iterations, $numBrackets = 1$, average of 50 runs



(e) 2000 Iterations, $numBrackets = 5$

Figure 2: Description of and Experimental Results for the Cross Function.

samples for that slope. This provided us with a *coverage score* for each slope, showing whether the technique had indeed concentrated its search points on the steeper slopes.

In each case our CROSSOVER procedure simply picked a random point along the line segment between the two parents, and the CREATE procedure generated a point at random uniformly within the space. DIST was simply the distance between the two points, and ASSESS returned the fitness at that point.

We chose three two-dimensional test cases for this problem. Each employed the sigmoid function,

$$\sigma(u, \beta) = \frac{1}{1 + e^{-\beta u}}$$

to create smooth transitions between roughly flat areas. The regions ran from -5 to 5 in both dimensions. In all cases, β was set to 5. The test cases were:

Cross: The Cross Function.

$$Cross(x, y) = \sigma(x, 5) + \sigma(y, 5)$$

This function is essentially sigmoid in both directions, creating a cross-like transition centered in the space. The function is shown in Figure 2(a), and the magnitude of its gradient is shown in Figure 2(b). There is very a small region in the exact center of the space which has a steeper slope than all other regions.

Rot: The Rotated, Tilted Cross Function.

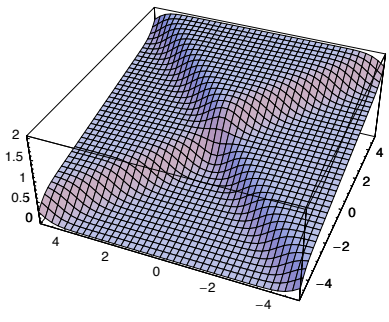
$$Rot(x, y) = \frac{1}{2}\sigma\left(\frac{x}{\sqrt{2}} - \frac{y}{\sqrt{2}}, 5\right) + \frac{1}{2}\sigma\left(\frac{x}{\sqrt{2}} + \frac{y}{\sqrt{2}}, 5\right) + \frac{x+5}{10}$$

This function rotates the cross function and adds a linear slope in one direction. Additionally, the magnitude of the sigmoids is reduced. The rotation is intended to move transitions off of dimensional boundaries, and the added slope and reduced magnitude is meant to complicate the task of finding non-zero slopes. The function is shown in Figure 3(a), and the magnitude of its gradient is shown in Figure 3(b). Again, there is very a small region in the center of the space which has a steeper slope than all other regions. Note that due to the tiltedness of the function, there are no regions with a slope less than 0.2.

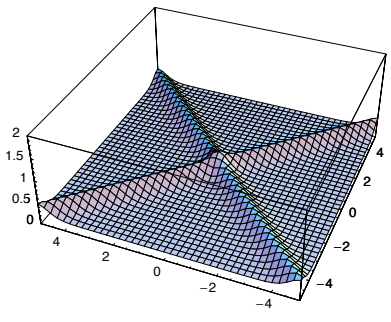
Circ: The Two-Circle Function.

$$Circ(x, y) = 1 + \sigma\left(\sqrt{x^2 + y^2} - 4, 5\right) - \sigma\left(\sqrt{(x+2)^2 + (y+2)^2} - 1, 5\right)$$

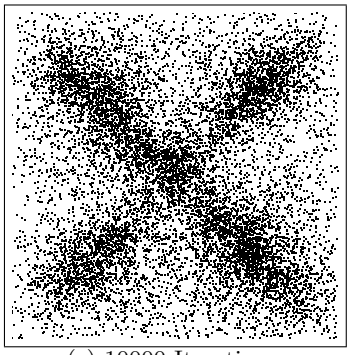
This function creates two circles, one within the other and inverted relative to one another. The inner circle is off-center in order to add some assymetry. The function is shown in Figure 4(a), and the magnitude of its gradient is shown in Figure 4(b).



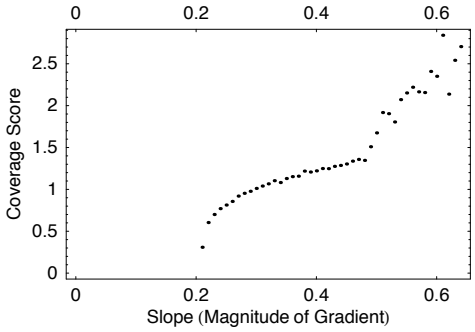
(a) Function $Rot(x, y)$



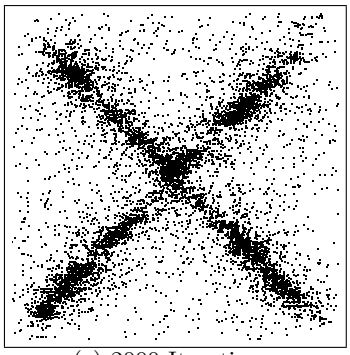
(b) Slope: $||\nabla Rot(x, y)||$



(c) 10000 Iterations, $numBrackets = 1$



(d) Coverage Score by Slope, 10000 iterations, $numBrackets = 1$, average of 50 runs



(e) 2000 Iterations, $numBrackets = 5$

Figure 3: Description of and Experimental Results for the Rotated, Tilted Cross Function (Rot).

4.1 Results

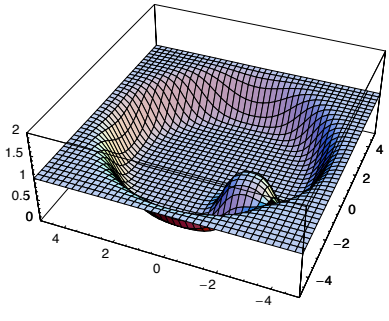
To compute the coverage score, we performed 50 runs of each test case and plotted the mean coverage score for each slope value. These runs were performed with 10000 iterations, $numBrackets = 1$, $initializationSize = 500$, $exploreProbability = 0.1$, $fitTourn = 10$, and $distTourn = 15$. This is a highly conservative run, with no bracketing at all; we felt it was best to establish the efficacy of the technique even at its weakest setting. We performed one additional run with these parameters to plot the result visually, and also one run with 2000 iterations but $numBrackets = 5$ to compare against it. This second run, which has approximately the same number of total samples, shows the effect of strong iterated bracketing on packing the samples much more densely along the steepest slopes.

One of the problems with the technique at present is that it has a difficult time sampling elements along the outer edges of the environment. The reason for this is that the probability that a child will be selected from near an edge is very low, because it requires that both parents be generated near that edge as well. We found that performing the metric distance tournament before the fitness tournament was important to counter this at least partially.

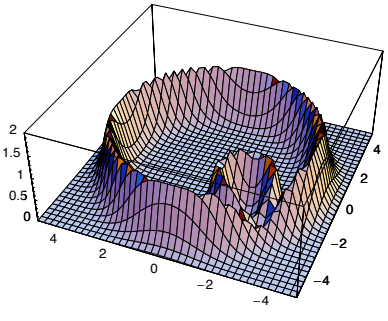
Cross: The Cross Function. As shown in Figure 2(c), the method is able to focus samples on the slopes of the function quite effectively. Figure 2(d) shows that as the slope increases, the density of samples increases monotonically. Slopes steeper than about 1.4 show a significant variance due to the very small number of patches in the environment at that steepness, and so are not statistically reliable. But the trend is very clear. Most of the environment has a slope of approximately 0, yet those samples had a coverage score of less than 1/6 that of slopes of 1.2 or so. Figure 2(e) shows the effect of bracketing: samples are much more closely packed along steep slopes in the environment.

From the visualization in Figures 2(c) and (e), we note two flaws in the technique. First, the visualization graphically shows the difficulty the method encounters in extending out to the edges, due to the reasons discussed earlier. Second, we note that slopes near (but not on) the center tend to have fewer samples than we would have expected. We are not certain why this occurs but believe that it is probably due to the tournament selection mechanism: after picking a parent in a "low" region somewhere near the center, it is fairly likely that its companion parent will be chosen in the "high" region diagonally opposite it rather than in the "medium" regions on either side, because the fitness differential is higher. Thus fewer children may be sampled along the edge transitions near the center and more will be sampled on the diagonal transition directly in the center itself.

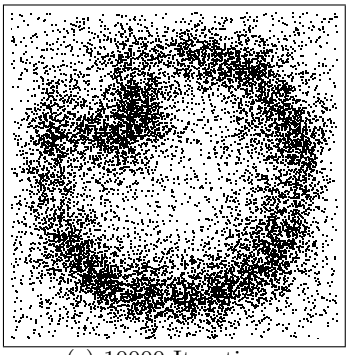
Rot: The Rotated, Tilted Cross Function. Recall that in this function, we rotated the cross, reduced its strength, and tilted it, in order to eliminate zero-slope regions, to make the "interesting areas" more difficult to discover, and



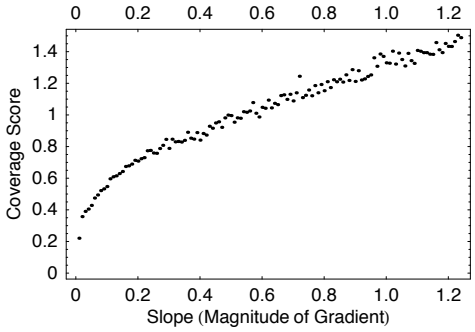
(a) Function $Circ(x, y)$



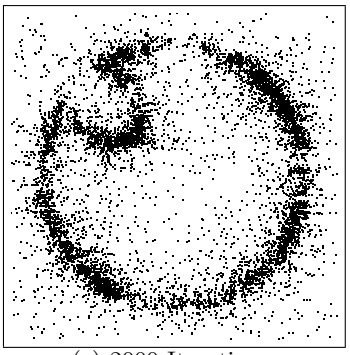
(b) Slope: $||\nabla Circ(x, y)||$



(c) 10000 Iterations, $numBrackets = 1$



(d) Coverage Score by Slope, 10000 iterations, $numBrackets = 1$, average of 50 runs



(e) 2000 Iterations, $numBrackets = 5$

Figure 4: Description of and Experimental Results for the Two-Circle Function (Circ).

to move the transitions off-dimension. The results are shown in Figures 3(c), (d), and (e).

As it turns out, the algorithm had no problem discovering the revised slope transitions, with very similar results to the *Cross* function. Again, beyond slopes of 0.5 the number of patches is too small and the sample variance cannot be trusted. As was the case for the Cross function, the Rot function proved difficult at the edges of the space. Note that in Figure 3(d) there are no slope patches at all until 0.2, hence no plot points.

Circ: The Two-Circle Function. In the last function there were no sloped regions in the center of the space, and an assymetry had been introduced. The results are shown in Figures 4(c), (d), and (e). Here again, the technique had little difficulty concentrating on the high-slope regions, producing coverage scores which grew with slope. This function has no small, high-slope patches, and so there are no small-sample issues at the extreme of the coverage score curve.

Once again, however, the four edges in the space were sampled poorly, even though there were no strong-slope regions in the center of the space.

5. DISCUSSION

This technique is new and there are a large number of ways that it could be improved and further analyzed. We discuss several issues and difficulties with the algorithm here.

We have so far only tested on two-dimensional spaces, largely to get a visual understanding of the technique. The obvious future direction is to determine how effective the method is in higher dimensions. At high dimensions these spaces get sparse very rapidly, and sampling such spaces can become difficult, much less sampling them in an adaptive fashion. One possible future approach is to perform dimensionality reduction techniques (Principal Components Analysis for example) to help reduce the sparsity of the environment and thus, ideally, the number of points to sample. Related to this is another problem: the method takes a while to build up enough samples to effectively adapt the sampling method. The technique seems to work well, but it does so more slowly than we'd like.

The algorithm tends to ignore points along the edges of the space. In our initial experiments we had swapped the order of the tournament selections (doing fitness first); and this produced a very strong tendency to avoid edges. Performing tournament selection on distance first helped alleviate this, but ultimately we will need a different procedure to select parent pairs. For example, if the first chosen point is close to an edge, we might increase the probability that another point along that edge will also be selected.

If a space has a consistent slope (such as in the *Rot* function), we note that the algorithm essentially ignores the mild constant slope permeating the search space. This is due to the use of tournament selection, which ignores candidate pairs' actual fitness differences and instead focuses on their relative ordering. But is this appropriate? Such a slope indicates that *something* is changing, after all. It appears

that the algorithm focuses samples not proportional to slope values but instead on those regions which have higher slope *relative* to their peers. This may or may not be desirable to the experimenter.

The algorithm also works well when there are large "uninteresting" spaces, but not necessarily when there are large numbers of "interesting" ones. In informal analysis, the algorithm appears to produce unfocused samples on functions such as two-dimensional sine-waves, $f(x, y) = sin(2\pi x) + sin(2\pi y)$, or similar functions such as Rastrigin $f(x, y) = x^2 + y^2 + a(1 - \cos(2\pi x)) + a(1 - \cos(2\pi y))$. Of course, these areas have few "uninteresting" areas to skip. The function complexity is high enough, and spread so widely throughout the space, that it's not clear if there really *is* an area that should be sampled less than others.

Last, this algorithm has not been compared against others: largely because we have failed to find any other algorithms which search for slopes in a multidimensional space. The only real competitor we have found is our own proposal to perform curve fitting in some fashion to the response surface of the function, and then sampling proportional to the slope of the surface. But one of the attractions our population-based method held was that it was essentially model-free, requiring no *a priori* knowledge of the space like curve-fitting would. Even so, comparing against a curve-fitting function would be useful in future work.

6. CONCLUSIONS

We introduced a novel solution to a heretofore little-studied problem: how to adaptively sample the space so as to focus on places in the space where the function output is changing. Our approach, a form of population-oriented iterated bracketing, is very effective on three initial test problems. But as discussed in the previous section, it is not yet clear whether this technique will scale, given the sparsity of high-dimensional spaces. Also, examining response surface modeling techniques may be a useful alternative approach.

One of the unusual side benefits of this method is that it shows how population-oriented methods may be used for tasks other than simple optimization. We think this work could be extended to other kinds of "non-optimization" search problems of interest to simulation designers: for example, finding locations where the output function is sensitive to one parameter but not to others; finding locations where one of several objectives changes but not others; or finding locations where the frequency of change is high.

7. ACKNOWLEDGMENTS

The authors wish to thank Jyh-Ming Lien for his assistance. The original idea behind this paper was born from a discussion with Dawn Parker, a multi-agent modeler.

8. REFERENCES

[1] N. M. Amato and Y. Wu. A randomized roadmap method for path and manipulation planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, 1996.

[2] S. Baluja. Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning. Technical Report CMU-CS-94-163, Carnegy Mellon University, Pittsburgh, PA, 1994.

[3] S. Ghosh and C. R. Rao, editors. *Design and Analysis of Experiments*, volume 13 of *Handbook of Statistics*. Elsevier Science, 1996.

[4] G. R. Harik, F. G. Lobo, and D. E. Goldberg. The compact genetic algorithm. In *Proceedings of the IEEE Conference on Evolutionary Computation*, 1998.

[5] D. Hsu, T. Jiang, J. Reif, and Z. Sun. The bridge test for sampling narrow passages with probabilistic roadmap planners. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2003.

[6] P. Larrañaga, R. Etxeberria, J. A. Lozano, and J. M. Peña. Optimization in continuous domains by learning and simulation of gaussian networks. In *Optimization By Building and Using Probabilistic*, pages 201–204, Las Vegas, Nevada, USA, 8 July 2000.

[7] P. Larrañaga, J. A. Lozano, and E. Bengoetxea. Estimation of distribution algorithms based on multivariate normal and gaussian networks. Technical Report KZZA-IK-1-01, Department of Computer Science and Artificial Intelligence, University of the Basque Country, Spain, 2001.

[8] M. McKay, R. Beckman, and W. Conover. A comparison of three methods for selecting values of input variables in the analysis of output from a compuer code. *Technometrics*, 21(2), 1979.

[9] D. C. Montgomery. *Design and Analysis of Experiments*. John Wiley, New-York, 2005.

[10] H. Mühlenbein. The equation for response to selection and its use for prediction. *Evolutionary Computation*, 5(3), 1997.

[11] M. Pelikan, D. E. Goldberg, and E. Cantú-Paz. BOA: The Bayesian optimization algorithm. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference GECCO-99*, volume I, pages 525–532, Orlando, FL, 1999. Morgan Kaufmann Publishers, San Fransisco, CA.

[12] S. Rudlof and M. Köppen. Stochastic hill climbing by vectors of normal distributions. In *Proceedings of the First Online Workshop on Soft Computing, WSC1*, 1996.

[13] G. G. Wang. Adaptive response surface method using inherited latin hypercube design points. *ASME Transactions, Journal of Mechanical Design*, 125(2):210–220, 2003.

[14] G. G. Wang, Z. Dong, and P. Aitchison. Adaptive response surface method a global optimization scheme for computation-intensive design problems. *Journal of Engineering Optimization*, 33(6):707–734, 2001.

[15] Y. S. Yeun, B. J. Kim, Y. S. Yang, and W. S. Ruy. Polynomial genetic programming for response surface modeling part 2: adaptive approximate models with probabilistic optimization problems. *Structural and Multidisciplinary Optimization*, 29(1):35–49, Jan. 2005.

[16] Y. S. Yeun, Y. S. Yang, W. S. Ruy, and B. J. Kim. Polynomial genetic programming for response surface modeling part 1: a methodology. *Structural and Multidisciplinary Optimization*, 29(1):19–34, Jan. 2005.