A Multiresolutional Estimated Gradient Architecture for Global Optimization

Megan Hazen, *Student Member, IEEE* and Maya R. Gupta, *Member, IEEE*

***Abstract*— In this paper we present a novel optimization algorithm that estimates gradients over regions to search for optima of a non-convex function on both a local and global scale. The proposed architecture is based on three concepts: using the memory of previously evaluated points, multiresolutional search, and the estimation of gradients at these different resolutions to direct the search. This multiresolution estimated gradient architecture (MEGA) shows promise to perform competitively when compared to standard global searches. Comparisons on the Rosenbrock, Griewank, and sinusoidal test functions show that MEGA can converge faster than particle swarm optimization, particularly as dimensionality of a problem increases.**

I. Introduction

We propose an architecture for global optimization of black-box objective functions. The focus is on optimizing functions which are relatively costly or time-consuming to evaluate. We discuss how our proposed algorithm relates to existing successful algorithms, focusing on the use of memory, gradient estimation, and multiresolutional search. The proposed algorithm combines these three properties and the goal of minimizing the number of parameters that must be tuned for each application.

The proposed architecture is described in pseudocode in Section III. The architecture description leaves many details open. A specific implementation of the details follows the pseudocode. The given implementation performed well over a number of test functions without the need for parameters to be tuned. Test results are summarized in Section IV. This implementation is compared to two particle swarm optimizers, and the performance is competitive in both convergence rate and number of function evaluations per run. The algorithms were compared on a number of objective functions with variable input dimensionality, and the MEGA performance is more consistent than the particle swarm optimizers as the dimensionality of the problem increases. This feature makes MEGA a promising choice for solving complex problems. A more complete discussion of the performance may be found in Section V.

II. Background

The search space we consider is some compact set $\mathcal{S} \subset \mathcal{R}^D$. The optimization problem is to find a global minimum x^* of an objective function $f(x)$ such that

$$x^* = \underset{x \in \mathcal{S}}{\operatorname{argmin}} f(x). \quad (1)$$

Megan Hazen is with the Applied Physics Laboratory, University of Washington, Seattle, WA 98195, USA (email: megan@apl.washington.edu).

Maya R. Gupta is with the Dept. of Electrical Engineering, University of Washington, Seattle, WA 98195, USA (email: gupta@ee.washington.edu).

Functions may also have local, or relative, minima, where the local minimum x_p^* of some region $\mathcal{S}_p \subset \mathcal{S}$ solves

$$x_p^* = \underset{x \in \mathcal{S}_p}{\operatorname{argmin}} f(x).$$

In many engineering problems the objective function is modeled as a black box; $f(x)$ can only be obtained for a specific x by running a program, taking measurements, or modeling a system. Functional information, such as an analytical gradient, is unobtainable, and the cost of each function evaluation may be high relative to the computation proposed to determine a candidate optimal point $\hat{x}$.

In the next subsections, we discuss three properties of successful algorithms: multiresolutional search, gradient estimation, and efficient use of memory. MEGA was designed to have these properties.

A. Multiresolutional Search

Successful global optimization methods hone in on local optima without becoming trapped in them. MEGA handles this by searching at multiple resolutions in parallel: local searches and a global search. The local and global searches attempt to follow the path of steepest descent, where the direction of steepest descent is defined over the local or global resolution. A different approach to multiresolutional search is enacted by simulated annealing with a cooling schedule. The cooling schedule creates a multiresolutional search over time; initially there is a random search on a global scale, but over time the searching becomes more local [1]. Another approach to multiresolutional search is the use of hybrid algorithms that combine some form of stochastic search for the global portion, and a more structured gradient descent type algorithm for the last, more local step [2], [3]. In generalized pattern search [4] a local search is added to the grid based pattern search to help refine the pattern search estimate. Particle swarm optimization changes each agent's search velocity as a random function of the distance between the current point and a local best, and the distance between the point and the global best [5]. This effects a random balance between searching locally and searching at a coarser resolution.

B. Gradient Descent

Traditionally, optimization algorithms calculate or estimate a gradient and follow the path of steepest descent, or adjust for curvature, as done in Newton-Raphson. For high-dimensional black-box objective functions it is not trivial to estimate the gradient, and many gradient-free methods have been developed [6], [7], [4]. Strictly gradient-free methods are able to converge to global optima, but it is our contention that moving downhill can direct the search more quickly towards the optimal area.

Many global optimizaton algorithms explicitly or implicitly estimate which direction is downhill at a given operating point. For example, finite difference methods require $2D$ extra function evaluations to estimate a gradient. Simultaneous Perturbation Stochastic Approximation (SPSA) uses a modified finite difference method that requires only two extra function evaluations per estimate, regardless of the dimension of the problem [1].

When the functional space has local minima or noise, the exact gradient, which is the instantaneous slope at some point, may not be that useful. To avoid local minima or ignore noise, one may be interested in estimating what direction constitutes "downhill" over a region. Thus, we will refer to the gradient over a region S_p as the vector β^* that best fits a hyperplane to the function over the region S_p, that is:

$$[\beta^*, \beta_0^*] = \underset{\beta, \beta_0}{\operatorname{argmin}} \int_{x \in S_p} (f(x) - \beta^T x - \beta_0)^2 dx. \qquad (2)$$

MEGA estimates gradients over regions of different sizes (that is, at different resolutions) based only on previous function evaluations.

C. Memory

Every time the objective function is evaluated more information about the function is obtained. Some search strategies collect this information and use it to guide future search paths. Tabu Search [8], in particular, does this explicitly by keeping a *memory* log to record promising and tabu (poorly performing) search locations. Other algorithms do this implicitly by basing new search steps on one or more of the previous iterations results. MEGA keeps a database of previously evaluated points, and bases each new gradient estimation on $N+1$ previously evaluated points. New points replace poorly performing previous points at the rate of one point per function evaluation. In essence this means that the MEGA algorithm remembers its search path for a number of iterations, using that information to direct the search. However, because good points are kept, the memory can extend back arbitrarily long to remember relatively good locations.

III. Algorithm Description

The MEGA algorithm is an iterative scheme where multiple new operating points are determined and evaluated in each iteration. Within the algorithm there are a few sub-processes that can be implemented in a variety of ways. The general structure for solving a D dimensional search problem is presented below in pseudo-code, then subsections discuss the details of the different sub-processes.

```
Sample initial points $\{x_i\}$ for $i = 1$ to $(D+1)^2$
Evaluate initial points $\{f(x_i)\}$
$x^* \leftarrow \operatorname{argmin}_{\{x_i,\ i=1 \text{ to } (D+1)^2\}} f(x_i)$
Build database with $\{x_i, f(x_i)\}$
while convergence criteria not met do
    Cluster the set of points $\{x_i\}$ into $(D+1)$ local neighborhoods
    for all n=1 to $(D+1)$ do
        Estimate gradient $\beta_n^*$ based on $\{x_i, f(x_i)\}$ in neighborhood $n$
        Calculate new operating point $x_n$ in direction $\beta_n^*$
        Evaluate new point $f(x_n)$
        Update database with $(x_n, f(x_n))$
        if $f(x_n) \le f(x^*)$ then
            $(x^*, f(x^*)) \leftarrow (x_n, f(x_n))$
        end if
    end for
    Estimate global gradient $\beta_g^*$ based on pairs $\{x_n, f(x_n)\}$
    Calculate new operating point $x_g$ in direction $\beta_g^*$
    Evaluate new point $f(x_g)$
    Update database with $(x_g, f(x_g))$
    if $f(x_g) \le f(x^*)$ then
        $(x^*, f(x^*)) \leftarrow (x_g, f(x_g))$
    end if
end while
```

A. Initialization

To initialize the MEGA algorithm a set of starting points $\{x_i\}$, $i = 1, \ldots, (D+1)^2$ are chosen randomly from a uniform distribution over the search space S. The objective function is evaluated at each of the starting points to form the initial sample pairs $\{x_i, f(x_i)\}$.

The number of points needed for the MEGA algorithm is dictated by the number of points needed to estimate the gradients. Linear regression is used for each gradient estimate, thus for a D dimensional objective function, a minimum of $D+1$ sample points are needed to create a numerically stable estimate. (Actually, the β_0 offset value is discarded, so the computation could be performed with only D points. This work, however, used the full regression fit with $D+1$ points.) Then, the new points generated by each local step are used to estimate a global gradient; a minimum of $D+1$ local estimates are required for a full rank estimate for the linear regression coefficients. This results in a total of $(D+1)^2$ initial points. Thus, the number of initial points scales with the square of the dimensionality of the problem, while the number of parallel searches scales linearly with the dimensionality.

B. Clustering

At each iteration of the algorithm the database is reclustered into $D+1$ neighborhoods. An agglomerative (bottom-up) average-linkage clustering of all the database samples separates the samples into $D+1$ mutually exclusive clusters. The clustering begins by considering each sample its own cluster. Then there are $(D+1)^2 - (D+1)$ iterations of the clustering algorithm, and at each iteration the two clusters that are closest are joined. We used an average-linkage implementation such that the distance between two clusters is defined as the average distance between elements in each cluster.

A number of global optimization techniques use an initial clustering of points to begin local searches or to define sub-domains for search [9]. In MEGA, the clustering happens every iteration and is a way to incorporate local memory into the gradient estimation for the local gradient descent steps. The MEGA clustering is a computationally-intensive step, particularly as the dimensionality increases. In preliminary experiments, only re-clustering every q iterations for varying values of q was tried. The results significantly deteriorated. That suggests that the clusters are intermixing between iterations, and that this intermixing is significant for good performance.

C. Gradient Estimation

The gradients over the regions defined by each neighborhood of previously evaluated points is estimated using standard linear regression to fit a hyperplane $f(x) = \beta^T x + \beta_0$ to the sample pairs $\{x_i, f(x_i)\}$ by minimizing the squared error. For example, let the nth neighborhood contain the sample pairs $\{x_i, f(x_i)\}$ if $i \in \mathcal{J}_n$. Then the estimated gradient is β^*, where

$$[\beta^*, \beta_0^*] = \operatorname*{argmin}_{\beta, \beta_0} \sum_{i \in \mathcal{J}_n} (f(x_i) - (\beta^T x_i + \beta_0))^2,$$

where β_0^* is an offset that does not form part of the gradient estimate.

The nth local gradient estimate is formed by fitting a hyperplane to only the sample pairs $\{x_i, f(x_i)\}$ in the nth neighborhood. The global gradient estimate is formed by fitting a hyperplane to the $D+1$ newly estimated local sample points $\{x_n, f(x_n)\}$.

Unlike some gradient-based optimization methods, such as finite-differences or SPSA, no new points are evaluated in order to estimate the gradient. Using previously evaluated points to do a least-squares hyperplane fit is an approach also used in first-order response surface methods [10].

The estimated gradient is used to direct the search. Traditionally x_{new} is calculated using the equation $x_{new} = x_{old} - \sigma\beta^*$, where σ is a step size parameter. In MEGA, the point from which to move, x_{old}, is not pre-defined. Instead, the step is taken from the centroid of all the points used to estimate the gradient. That is,

$$x_{new} = \frac{\sum_{i \in \mathcal{J}_n} x_i}{\|\mathcal{J}_n\|} - \sigma\beta_n^*,$$

where $\|\mathcal{J}_n\|$ denotes the cardinality of the set $\mathcal{J}_n$.

The least-squares regression coefficient vector β^* has a closed form solution:

$$[\beta^* \beta_0^*]^T = (X^T X)^{-1} X^T y, \tag{3}$$

where X is a matrix with the sample point x_i in the ith row, and Y is a vector with $f(x_i)$ in the ith row, and the i+1th row of X and Y is all ones. The $(D+1) \times (D+1)$ matrix $X^T X$ must be invertible in order to form the gradient estimate. Thus, $D+1$ linearly independent samples points are needed. Before calculating each gradient estimate, a check is made to determine if there are enough linearly independent sample points. If not, another point from the database is randomly chosen and added to the estimation. Randomly chosen points from the database are added one-by-one until enough linearly independent samples are available for a numerically robust matrix inversion.

D. Database Management

MEGA maintains a database of previously-evaluated points which serves as the memory for the algorithm. Not all previously-evaluated points are kept in memory. At any given time there are $(D+1)^2$ points are in the database, one of which is noted to be the current best operating point: x^*. The current best operating point is replaced by some other point $\tilde{x}$ if $f(\tilde{x})$ is evaluated and $f(\tilde{x}) < f(x^*)$.

The database is initialized at the start of the optimization with $(D+1)^2$ randomly drawn pairs $\{x_i, f(x_i)\}$. Each time a new operating point $(x_{new}, f(x_{new}))$ is evaluated the database is updated to include that point. When a new point is added to the database, some point is removed. In this way the size of the database is maintained. Every time the nth neighborhood adds a point, the worst point (before the new addition) in the nth neighborhood cluster is removed. Every time the new global point is added, the worst point in the entire database (before the new addition) is removed.

The replacement of the worst point in the database, the requirement that gradient estimates be based on $D+1$ linearly independent samples, and the frequent re-clustering are the three aspects that in combination limit the possibility of spending infinite iterations on a local search around a relatively poor local optimum.

At all times, the database contains $(D+1)^2$ active points, $D+2$ of which are replaced each iteration, allowing memory to persist over many iterations. Each new point x_{new} is added regardless of $f(x_{new})$. If x_{new} is a relatively poor performer, then this information will inform the next estimated gradient to direct the search away from x_{new}.

E. Parameter Settings

Most optimization algorithms have some parameters that can be tuned to improve the algorithm's performance on a specific problem. In the MEGA design an effort was made to minimize any free parameters, and to provide defensible reasons for any parameter choices. Ideally, an algorithm would automatically and optimally adjust its own parameters to a new objective function.

The first MEGA parameter is the number of random sample points for the initialization. We choose the minimum number of points required for the full linear regression fit, $(D+1)^2$, as detailed above in the initialization subsection. Thus this parameter is coupled to the decision to use $D+1$ local searches. The number of local searches was chosen to provide the minimum number of local new points to create a full rank regression estimate for the global search. It should be noted that this results in a number of active points that is directly related to the dimensionality of the problem, which the authors hypothesize is necessary to efficiently search the space.

Another set of parameters is the starting value and decay schedule for the step size σ. If the step size is too large new points will be located seemingly at random, if σ is too small the new points will not be significantly different from existing points. The starting σ is set to be half the length of the longest edge of the hyperrectangular space. This start value ensures that early iterations of the algorithm form a coarse painting of the space: any given step can move entirely from one quadrant of the search area to another one. The decay schedule is a monotonic geometric decay schedule,

$$\sigma_{j+1} = \sigma_j \times 0.99,$$

where j indexes the iteration of the MEGA algorithm.

F. Boundary Conditions

The MEGA algorithm has been designed to work with black-box type objective functions. The search space for these functions is usually constrained, as the input values have practical ranges. In this work it has been assumed that all constraints are pre-determined limits on individual dimensions, so that the search space S is a hyper-rectangle. Hyper-rectangles have deep pockets in high-dimensional spaces, or equivalently, much of the volume of the space is in the corners. This can be understood by considering a hypersphere of radius 1 centered inside a hypercube of edge length 2. The volume of the hypersphere decays to zero as the dimension increases. The volume of the hypercube grows without limit. The hypersphere touches the edges of the hypercube for any given dimension. Thus, the volume of the search space is predominantly in the corners of the hypercube as the dimension grows. Related to this property, uniformly random sample points are likely to be near the edge of the search space. A consequence is that optimization algorithms searching the space by steepest descent are likely to have many steps that land outside the search space. (For more details on the curse of dimensionality see [11].)

In early incarnations of this work the constraints were enforced by limiting the values the input vector could take on, and vector values violating the constraints were mapped to the closest boundary point along the estimated gradient. However, this method could cause sample points to cluster at the edge of a boundary. A cluster of such points would not be linearly independent, leading to difficulties in estimating gradients. This results in redundant searching in the best of cases.

For this reason, the boundaries are instead handled using slack variables [12], such that there is a penalty applied for new points that fall beyond the boundaries of the search space. Let b_l and b_u be D dimensional vectors specifying the hyper-rectangular boundary limits in each dimension. Then, if a new operating point x falls outside the limits of the search space S, MEGA gives x the value

$$\hat{f}(x) = f(x) + \sum_{d=1}^{D} e^{\|\delta_d\|}, \tag{4}$$

where δ_d is the distance x is outside the boundary in the dth dimension.

IV. Test Suite and Results

Three standard test functions [13] were chosen to stress different aspects of an optimizer. The standard Rosenbrock function is strictly convex, but considered challenging for algorithms using steepest descent. The sinusoidal function is non-convex, and has no global trend towards a minimum. The Griewank function is bowl-shaped but with many local minima within the search space.

A popular and powerful global optimization algorithm is particle swarm optimization [14]. The particle swarm optimization algorithm has similarities to MEGA in that they are both population based algorithms, and both use some estimate of a down hill direction to calculate the next step. Therefore, a reasonable goal for the MEGA algorithm is to match the performance of a particle swarm optimization algorithm. Recently, Schutte and Groenwold compared a number of particle swarm optimization variants [5]. MEGA is compared to one of the two variants they recommend based on their study, namely particle swarm optimization with constriction (CPSO), in which the velocity is multiplied by a constriction factor. To ensure a tuned performance, the algorithm was implemented using the parameters recommended by Schutte et al. Test results are also shown for a traditional particle swarm optimization implementation in which the velocity has no constriction multiplier. We refer to this second particle swarm optimization simply as PSO. Our PSO has equivalent velocity weight parameters as CPSO, but a different number of agents: the CPSO implementation uses 25 agents for each test run, while the PSO implementation uses $(D + 1)^2$ agents. This number of agents was chosen to be equal to the number of active sample points in any given iteration of the MEGA algorithm.

Here, as in Schutte's tests [5], a method is said to have *converged* for a function $f(x)$ if the search has discovered some $\bar{x}$ such that $f(\bar{x}) - f(x^*) \leq \epsilon$, where x^* is defined as in (1), and $\epsilon = .001$. Similarly, each algorithm was allowed to run for a maximum of $1000D$ function evaluations. The results are presented in terms of percent convergence and average number of function evaluations (averaged only over the converged instances). These values are plotted as a function of dimension for each of the test functions.

The Rosenbrock function is one of the best known test functions. Its long valley makes it particularly hard for algorithms to find the precise minimal location. The results of running all three test algorithms on this function are shown in Figure 1. The results for the sinusoidal test function are shown in Figure 2, and for the Griewank function in Figure 3. Of the test functions the Griewank function posed the most

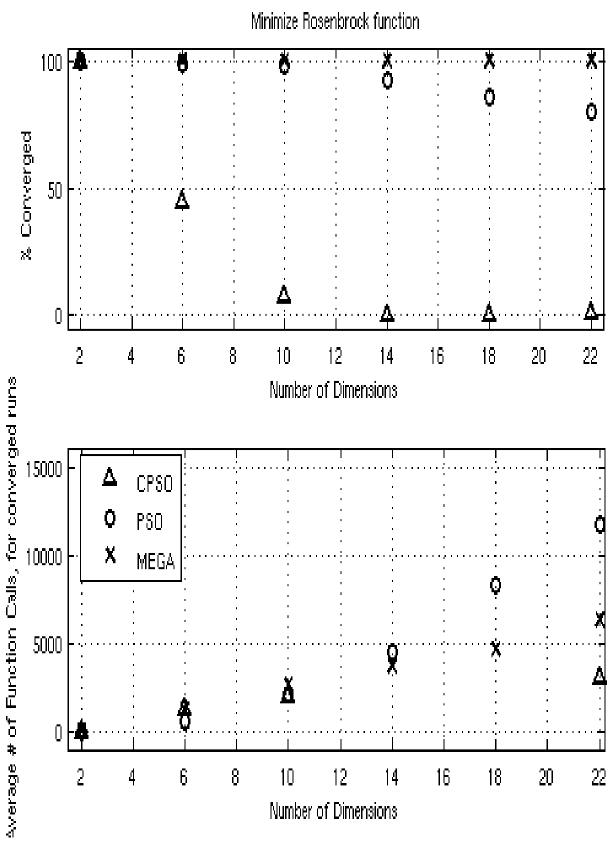

Fig. 1. Statistics for minimizing the Rosenbrock function. Top: percentage of the 100 test runs that converged. Bottom: average number of function evaluations for the converged runs. (Runs which did not converge are not included in this average.)

difficulty, possibly due to the high number of local minima within the search space.

Examining the plots we see that all three algorithms perform similarly in low-dimensional spaces. For the sinusoidal and Rosenbrock functions all algorithms converge frequently with relatively few function evaluations. The Griewank function poses a bigger challenge and all the algorithms struggle with it. An interesting aspect of the MEGA design is that it specifically scales with dimension, and converges consistently even when the dimensionality increases. The performance of the other two algorithms, however, drops off as the number of dimensions is increased.

It is also interesting to look at the number of function evaluations needed for convergence. MEGA requires fewer function evaluations for the Rosenbrock function, presumably because the convex space is well suited to gradient direction search. MEGA also required fewer function evaluations as the number of dimensions is increased. In some cases the PSO variants required fewer function evaluations at low dimensions.

While this testing showed that MEGA offers consistent performance at relatively low dimensionality, further testing was performed to determine if MEGA could maintain its consistent convergence rates at even higher dimensions. Figure 4 compares the performance for optimizing the sinusoidal function over 30, 40, and 50 dimensions, where the maximum allowed function evaluations was again $1000D$. MEGA converged 100% of the time for 100 runs in each dimension. The results are a function of the maximum number of function evaluations allowed and the convergence criteria threshold ϵ. A more complete picture is had by considering Figures 5 and 6, which show histograms of the final values of the 100 runs for 40 dimensions and 50 dimensions. These figures show that the results would not vary with moderate changes in ϵ. The number of PSO and CPSO runs that ended at quite high function values suggests that it would take significantly more function evaluations to converge to the precise optimum.

V. Discussion

This paper has presented a novel optimization algorithm that has shown promise to compete with other successful modern optimizers. The testing shows that MEGA performs consistently as the dimensionality of the search space increases, and frequently converges with fewer function evalu-

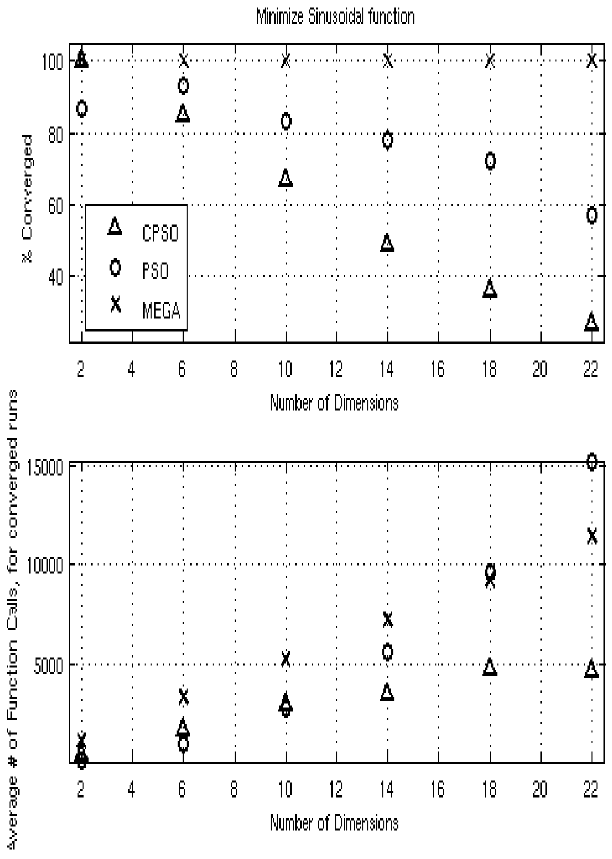

Fig. 2. Statistics for minimizing the sinusoidal function. Top: percentage of the 100 test runs that converged. Bottom: average number of function evaluations for the converged runs. (Runs which did not converge are not included in this average.)

ations than its competitors. However, there are some notable open issues that will be addressed in future efforts.

A. *Step Size Parameter*

During the development of the MEGA algorithm the authors tried to minimize the number of parameters that needed to be adjusted by the user for each new objective function. The parameters that are necessary should, when possible, be automatically set based on functional information.

Controlling the step size is an essential part of the optimization algorithm. Many algorithms, MEGA included, allow the step size to vary over the course of the run so that early steps may find a promising region of attraction while later steps find a more precise location. Currently the initial step size is based on the maximum size of the search space so it scales automatically to the current, but the decay schedule is somewhat arbitrary. Ideally the decay schedule would be based on the performance of the search to date.

In the MEGA algorithm it should be useful to scale the step size to the resolution of the current search. Currently there is one step size value for movement along either a neighborhood or global estimated gradient. An improvement might be to base the step size on the range of the points going into the current step calculation. This will result in shorter steps for the local searches, while the global search will still be allowed to extend across the entire search space. Additionally, basing the step size on the current range

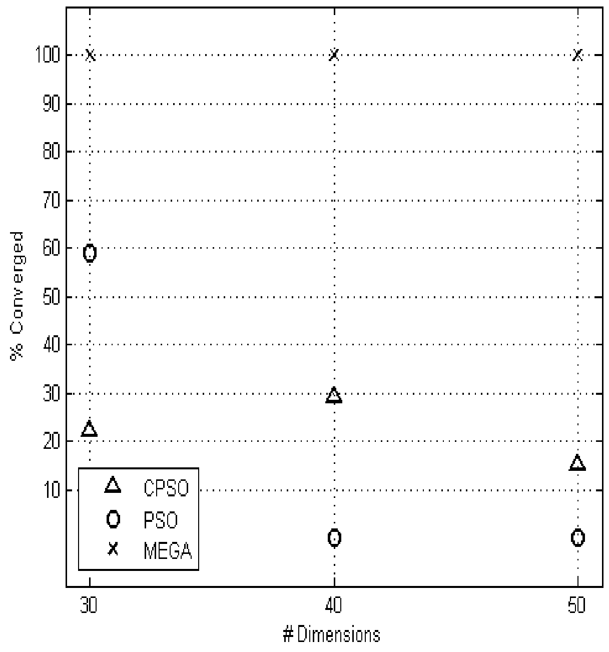

Fig. 4. Percentage of the 100 test runs that converged for finding the optimum of the sinusoidal function over higher dimensions.

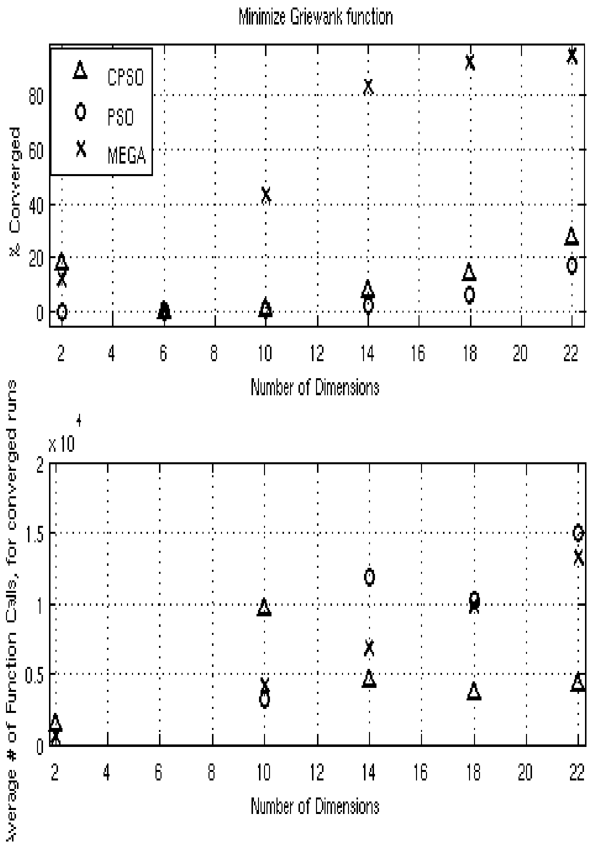

Fig. 3. Statistics for minimizing the Griewank function. Top: percentage of the 100 test runs that converged. Bottom: average number of function evaluations for the converged runs. (Runs which did not converge are not included in this average.)

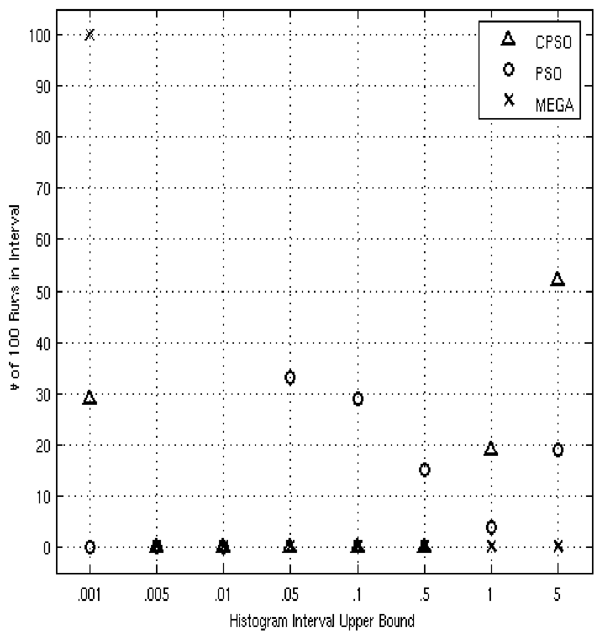

Fig. 5. Histogram of the final values of the 100 runs for 40 dimensions.

of searching points would remove the need for an extra decay schedule, as the step size would diminish as the searching points converged to a more local neighborhood. We hypothesize that improving the automatic adaptation of the step-size parameter will result in a more efficient algorithm.

B. Maintaining Rank

A difficult challenge faced by the MEGA algorithm is ensuring that each gradient estimation is done over D linearly independent points. When there are not enough points in the neighborhood, additional effort must be expended to add points from the database to the pool. Additionally, if the criteria is minimally met the regression estimate may be poor due to ill-conditioned matrix inversion. This can occur when there is a very tight cluster of points. In this case the points are redundant because they are basically co-located. All of these issues cause inefficiency in the search, and fixing them is expected to further reduce the number of function evaluations needed for convergence.

In future work methods for ensuring a relevant supply of linearly independent points will be sought. Currently points from the database are added to the regression set until the input matrix is of rank $D + 1$. These points are chosen randomly, and it is likely that a more careful method of choosing points would result in the need to add fewer points overall. One criteria for adding additional points is to add points based on their distance from the neighborhood; this may result in more accurate gradient estimations for the neighborhood.

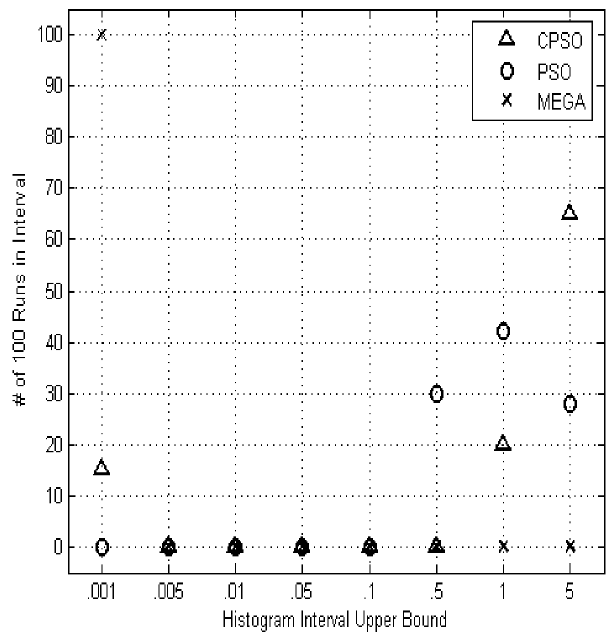

Fig. 6. Histogram of the final values of the 100 runs for 50 dimensions.

C. Computational Complexity

It should be noted that, while MEGA performs competitively when the metric is the number of function evaluations, it is relatively slow when the metric is overall time. For problems in which the function to be optimized takes significant time to calculate, it may be worthwhile to spend more time choosing the next point to evaluate. At the one extreme is pure random search: when function evaluations are free there is no need for intelligent global optimization. MEGA is at the other extreme, function evaluations must be fairly expensive to justify the computational complexity. However, the specific implementation of the general MEGA architecture can be modified to run faster. In particular, the time could be cut by employing faster methods for finding local neighborhoods (through a different clustering mechanism, or some other means), and faster methods for estimation of the gradients over regions.

D. Particle Swarm Optimization Discussion

This paper is not designed to recommend the best ways to use the particle swarm optimization algorithm and its variants. The implementations tested here are specifically matched to recommendations from previous work. However, a few interesting points can be made. CPSO was originally proposed in [15], based on an algebraic analysis of the way individual particles converge to stable positions. The parameters used were recommended initially in that work and further subjected to experiments in [5].

We note that CPSO is equivalent to the basic particle swarm optimization implementation, with parameters that are weighted. The second particle swarm optimizer tested here, PSO, uses parameter settings equivalent to those in the CPSO implementation. However, for the second implementation the authors also modified the number of particles to match the cardinality of the number of active points in the MEGA algorithm, specifically $(D+1)^2$. PSO's particles, and MEGA's active points, act as the memory of the optimization.

The plots show that CPSO and PSO converge at similar rates for low dimensions, but the PSO performs more reliably as the number of dimensions increases. This implies that increasing the number of agents with increasing dimension does improve the convergence of a basic PSO algorithm. At around forty dimensions we note that PSO begins to fail to converge, while CPSO maintains its low rate of convergence.

These results should come as no surprise. One common failure mode of particle swarm optimization algorithms is early convergence to local optima. If there are not enough particles searching the space the area of attraction for the real optimum may never be encountered. Increasing the number of particles is a useful way to ensure that the best area of attraction is found. However, many recommendations are to keep the number of particles at a low value (20-30) because using more particles results in a higher number of function evaluations needed for convergence. In the highest dimension cases tested here (40-50 dimensions) the PSO implementation is not given enough function evaluations to converge, while the CPSO implementation can still converge occasionally in the allotted time. The histograms for 40 and 50 dimensions (shown in Figures 5 and 6) support this hypothesis because the final non-converged values for the PSO implementation are on average smaller than the final non-converged values for the CPSO implementation. This suggests that given infinite time the 100 PSO runs would have converged on average faster than the 100 CPSO runs. A recommendation based on this hypothesis is to increase the number of particles in high dimensional problems to ensure convergence, but to keep the number of particles low if a (possibly suboptimal) answer is needed in fewer function evaluations.

VI. Conclusion

In this paper the ideas of memory usage, multiresolutional searching, and gradient estimation were combined into a new global optimization architecture. Early testing of a specific implementation shows competitive performance when compared to particle swarm optimization. MEGA appears to be particularly good at adjusting to an increasing dimensionality of the search space. The implementation presented has a number of detailed processes (clustering, regression fitting, and step-size scaling) that will be improved in future work.

This future work includes the step size and rank issues discussed above. It should also be possible to improve many of the aspects of the MEGA algorithm. Replacing the regression and clustering algorithms with more efficient implementations will reduce the computational complexity of the optimization scheme. The database management may also be improved by examining what points are added to the database and how many points are retained.

Acknowledgment

This work was supported in part by the Applied Physics Lab. The authors would like to thank Warren Fox for helpful discussions.

References

[1] J.C. Spall, *Introduction to Stochastic Search and Optimization*, Wiley-Interscience, Hoboken, NJ, 2003.
[2] K.F.C. Yiu, Y. Liu, and K.L. Teo, "A hybrid descent method for global optimization," *Journal of Global Optimization*, vol. 28, pp. 229–238, 2004.
[3] O. Polgar, M. Fried, and I. Barsony, "A combined topographical search strategy with ellipsometric application," *Journal of Global Optimization*, vol. 19, pp. 383–401, 2001.
[4] C. Audet and J.E. Dennis Jr., "Analysis of generalized pattern searches," *SIAM Journal on Optimization*, vol. 13, no. 3, pp. 889–903, 2003.
[5] J.F. Schutte and A.A. Groenwold, "A study of global optimization using particle swarm," *Journal of Global Optimization*, vol. 31, pp. 93–108, 2005.
[6] D.A. Pierre, *Optimization Theory With Applications*, Dover Publications, Inc., New York, NY, 1986.
[7] Z.B. Zabinsky, *Stochastic Adaptive Search for Global Optimization*, Kluwer Academic Publishers, Norwell, MA, 2003.
[8] F. Glover and M. Laguna, *Handbook of Applied Optimization*, chapter 3.6.7: Tabu Search, pp. 194–208, Oxford University Press, New York, NY, 2002.
[9] D. Solomatine, "Two strategies of adaptive cluster covering with descent and their comparison to other algorithms," *Journal of Global Optimization*, vol. 14, pp. 55–79, 1999.
[10] R.H. Myers and D.C. Montgomery, *Response Surface Methodology: Process and Product Optimization Using Designed Experiements*, Wiley-Interscience, USA, 2002.
[11] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, Springer-Verlag, New York, 2001.
[12] P.Y. Papalambros and D.J. Wilde, *Principles of Optimal Design*, Cambridge University Press, Cambridge, UK, 1988.
[13] X. Yao, Y. Liu, and G.M. Lin, "Evolutionary programming made faster," *IEEE Transactions on Evolutionary Programming*, vol. 3, pp. 82–102, July 1999.
[14] R. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in *Sixth International Symposium on Micro Machine and Human Science*. October 1995, pp. 39–43, IEEE.
[15] M. Clerc and J. Kennedy, "The particle swarm - explosion, stability, and convergence in a multidimensional complex space," *IEEE Trans. on Evolutionary Computation*, vol. 6, no. 1, pp. 58–72, 2002.