Particle Swarm Optimization

Joy Ding and Drew Robb
Harvard College
(Dated: December 11, 2009)

ABSTRACT Particle Swarm Optimization (PSO) is an evolutionary algorithm based off of swarm intelligence and is used as a stochastic optimization technique. One major characteristic of PSO is many configuration parameters, which allow the algorithm to be adjusted to various problem landscapes. In the first half of this paper, we empirically benchmark performance of different parameterizations of PSO in a fractal-like test environment for convergence and robustness against local optima. Experiments were done with most salient parameters: dampening factor, neighborhood size and dimensionality. We evaluate a proposed technique of eliminating the known problem of dimensional collapse and position bias, as well as propose a novel optimization technique for restarting particles which have prematurely converged to increase robustness against local optima. PSO was then successfully applied to an image matching algorithm called Evolisa, using a combination of hill-climbing and PSO. PSOlisa also employs several optimization techniques to improve convergence efficiency.

I. INTRODUCTION

Particle Swarm Optimization (PSO) was first introduced as an evolutionary stochastic optimization technique for nonlinear functions by Kennedy and Eberhart in 1995. [1] Since then, it has proven to be an efficient optimizer applicable to a multitude of global optimization problems. Like other evolutionary techniques such as Genetic Algorithms (GA), PSO is initialized with a population of random solutions and updates the population every generation to search for the optimum.

However, instead of traditional evolutionary operators such as mutation or crossover, PSO is inspired by the advantage of cooperative and social behavior seen in swarm intelligence. In search of the global minimum, particles in the population fly through the search space, following particles with the current best optimum found. Each particle keeps track of its personal best fitness ever achieved, as well as the fitness value of the current solution. The swarm also tracks the best value ever achieved by the swarm as a whole, as well as in some implementations, the localized neighborhood of particles surrounding the particle. The particles move in the space, according the best values of the particle itself, its neighborhood, and the entire swarm. However, the analogies to swarms and particle paths are slightly misconceived. The particles in each iteration undergo a randomized acceleration and typically maintain a large velocity. A more accurate understanding of PSO requires realizing that the swarm dynamics simply create an intelligent, adaptive probability distribution for function evaluations that efficiently investigates promising regions.

Part of what makes PSO a particularly attractive algorithm is its multi-dimensional parameterization, which allows the algorithm to be fine-tuned to various problem requirements. The first half of this paper will thoroughly investigate the effects of parameterization on PSO over a series of controlled problem landscapes generated via a sinusoidal function. We then apply our analysis of PSO to an image-matching problem called EvoLisa in which a set of polygons mutates over generations to match a reference image. [2] We then propose several optimizations to encourage convergence and robustness in the algorithm.

II. IMPLEMENTATION

We implemented a pure PSO sandbox enviroment and our PSO based version of the Evolisa algorithm on GPGPU hardwave with CUDA, using PyCUDA[3] as a wrapper. The purpose of using special hardware was a linked final project for a systems class (CS264). High level instructions and the generation of randomized data was done in Python, and the bulk of the computations were done on the GPU, using PyCUDA as the interface. The NVidia GTX260 device we used is capable of 715 GFLOPS, which out classes any CPU. We estimate that using the GPU gave us a 1-2 order of magnitude speedup, which was extremely helpful for running large simulations interactively.

III. PARTICLE SWARM OPTIMIZATION

A. Algorithm

The problem domain is an n dimensional continuous (using single precision floating point numbers) space. In the problems we study, this space is the unit cube $U[0,1]^n$ (U being the uniform distribution operator). There are s particles initialized to random positions (drawn from $U[0,1]^n$) and random velocities (drawn from $U[-1,1]^n$). All particles undergo synchronous updates. The particles evaluate the objective function each iteration and check it against two high water marks, the particle's own best function evaluation (optima) and the best optima amongst a set of neighbor particles. The neighbors are defined by a directed graph on the s particles. This graph defines the propagation of the neighborhood optima. We used the next-k topology. In this topology, particle i is connected to particles $i+1, ..., i+k \mod s$.

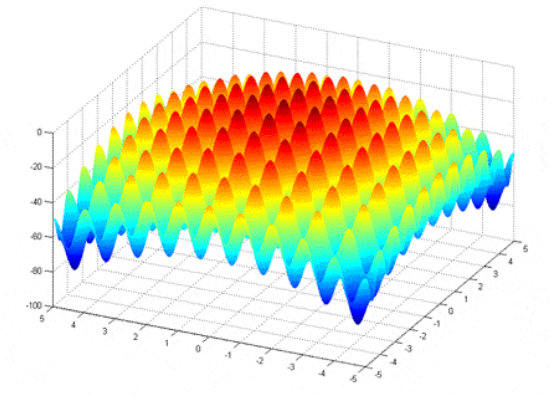

FIG. 1: Rastrigin's function is a standard benchmark for PSO and other optimizers. It is seen as a challenging function because of the many regularly spaced local maxima.

At each update, the particle at position $\mathbf{x}$ incurs an acceleration $\mathbf{a}$ defined in each dimension i by:

$$a_i = k_l U[0,1](lb_i - x_i) + k_n U[0,1](nb_i - x_i) \qquad (1)$$

where **lb** is the local best, **nb** the neighborhood best, and k_1, k_n are the spring constants, typically equal to 0.7. After some number of iterations, the maximum amongst the local optima is returned as the result of the optimization.

A particle also loses some energy, determined by a dampening parameter d, so that $\mathbf{v} \leftarrow d\mathbf{v}$ prior to acceleration. Typically d is slightly less than 1.

Some form of boundary conditions are necessary in most problems. Generally, it is a bad idea to use hard or soft boundary conditions to force the particle into the domain. This can bias particle position near corner states. For our implementation, we used free boundary conditions, but modified the objective function in a rational way for states outside the problem domain. For the PSO sandbox, we simply set the objective function to $-\infty$ outside the boundaries.

A second type of boundary condition guards against diverging particle energies. To implement this, we used a particle threshold. The threshold was randomized at every iteration, as this was a recommended technique in the literature.[4] The randomization was on the scale of the entire domain, as to not effect all but the most troublesome particles.

Program 1 PSO algorithm

```
intialize population of S particles
until (terminating condition):
    across all particles:
        integrate position
        evaluate objective function
        update local, neighbor and global bests
```

B. Test Environment

Typical PSO analysis in the literature relies upon non-random test functions, with predetermined global optima. For example, the Rastrigin function (Fig. 1)[5]. We felt that an analysis of PSO on artificial functions wouldn't be robust against real problems where the objective function is dependent on the input data. We use the following family of randomized fractal-like test functions of a n dimensional space with position vector $\mathbf{r}$:

$$F(\mathbf{r}) = \sum_{i=0}^{i=m} \sin\left(o_i + c_i(U[0,1]^n \cdot \hat{\mathbf{k}}_i)\right) \qquad (2)$$

where the normalized wave vector $\hat{\mathbf{k}}_i$ has a direction drawn from a uniform distribution, with a magnitude c_i drawn from the following probability distribution distribution for $x > 0$:

$$c_i = M(1 + Dx^{-F}) \qquad (3)$$

where M is a constant that determines the lowest frequency mode, and thus the number of maxima. We expect that the number of maxima in n dimensions is approximately M^n. The o_i term is simply a randomized offset chosen for each mode so that the 0 coordinate will not evaluate to 0. The parameters F and D determine how likely high frequency modes occur. D and F together determine the fractalization of the function. We wanted the function to have a global maximum determined by the lowest frequency modes, but to have local maxima appear at many different scales. The two parameters D and F allowed us to find such a distribution of frequencies. An overly fractal function is not tractable for PSO. We want the probability of an even greater optima being found to be higher around the optima that currently exist, since PSO directs particles towards optima. Fig. 2 shows an examination of the potential 2d landscapes generated from these equations. Fig. 3 shows the corresponding convergence distributions of these landscapes.

C. Parameter Sweeping

Unlike simple search algorithms, PSO has many parameters that can greatly characterize performance. We conducted many parameter sweeps and judged the algorithms performance on several of our randomized fractal-like test landscapes. This study interested us for two reasons. First, studying PSO by itself is interesting. Second, we wanted to use PSO efficiently in order to implement our version of EvoLisa. It should be noted that all of our examples represent just a 2 or 3d slice of possible parameter configurations. The ideal settings for one parameter almost certainly have a complex dependence on all the others.

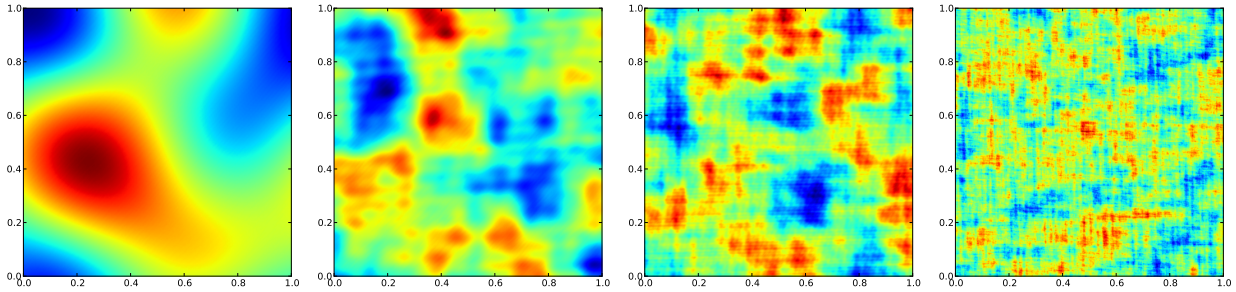

FIG. 2: For all landscapes, $M = 1$. Left: No fractalization, and $F = \infty$, so we expect a single maxima and no higher frequency terms. Second: $F = 4, D = 4$ the fractalization is not strong enough at high frequencies. Third: $F = 2, D = 3$ a good balance of parameters, creating fractal features at many scales that don't dominate features created by the lowest frequency modes. Right: $F = 1, D = 4$ The Fractalization occurs too strongly at high frequencies, dominating the low frequency features.

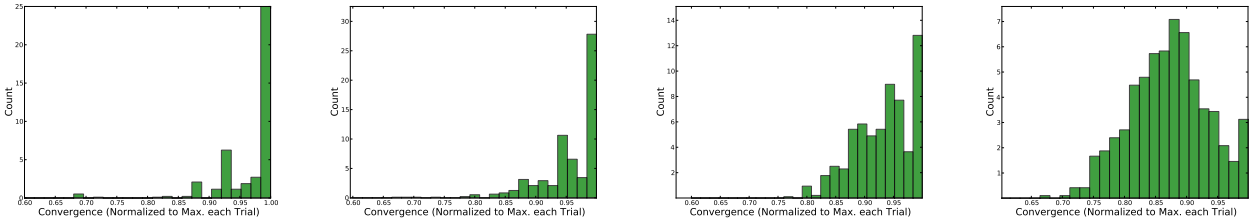

FIG. 3: Here we test performance of the corresponding landscapes of Fig. 2 generalized to $n = 6$ dimensions. We generate a landscape, run PSO 20 times and normalize all results based on the highest maximum found in and trial. A large histogram value in the furthest right bin indicates a high probability of trials finding the global maximum. With sufficient landscape complexity, each trial converges upon a different optima.

1. *Dampening Factor*

Each particle's velocity is multiplied by the dampening factor d at each iteration. This lowers particle energy, and allows for a faster collapse of the swarm towards the known optima. We can think of the dampening factor as a cooling parameter. We conducted a sweep of reasonable dampening factors, running 20 runs of PSO for each candidate factor on the same function landscape (Fig. 4). We found that a dampening factor less than 1 greatly increases the rate of convergence, however a dampening factor of exactly 1 results in good long term convergence. A dampening factor of greater than 1 results in extremely slow convergence. Closer examination of dampening factors progressing geometrically towards 1 showed that there is roughly a relation determining the ideal dampening parameter in terms of the number of iterations I:

$$(1 - d)I = C. \tag{4}$$

That is, when d and I are optimized, this expression is constant. This relation makes sense if we assume that upon the final iteration, and optimized dampening parameter would have the particles have the same minimum energy necessary to explore the smallest features of the state space.

2. *Neighborhood Size*

The neighborhood size used in PSO is another important parameter. More generally, the neighborhood topology is also important. This topology is easy to implement, and has a simple parameter that allows variation in the neighborhoods. When a particle finds a new optima, on the next iteration, k total particles will now be directed in its direction. The quantity s/k, where s is the total number of particles, is another important quantity. It will take s/k iterations for the entire swarm to learn of the new optima. Fig. 5 shows a table of the results of a sweep over swarm and neighborhood sizes using the next k topology, giving convergence values normalized to the best convergence, and averaged over many trials.

3. *Spring Constants*

Surprisingly, the spring constants that determine the acceleration of particles towards the optima had very little bearing on the efficiency of PSO. Most resources recommended using constants anywhere from 0.7-2.0. We found that this was the correct window, but fine tuning the values offed negligible benefits (Fig. 6).

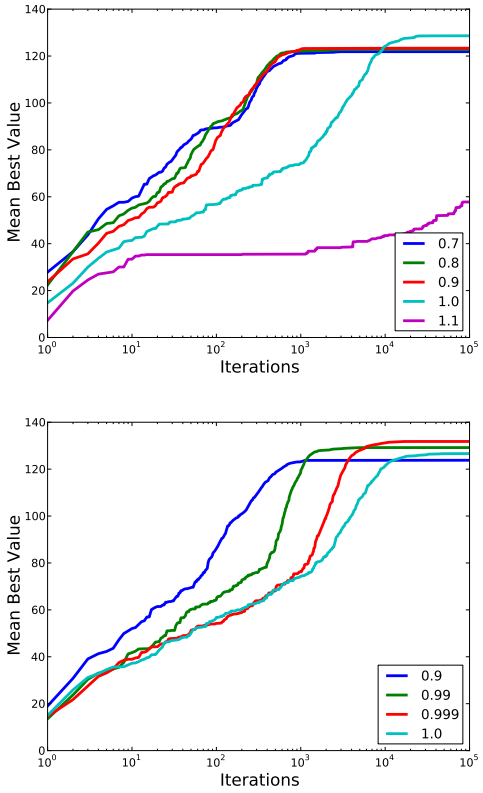

FIG. 4: Mean convergence, varying the dampening factor. Note the large number of iterations (10^5), which is much higher than ideal. The first figure shows a wide spectrum of factors, while the second shows a logarithmic progression of dampening factors. Note that in the second figure, each higher factor converges slower and better than the previous.

	8	16	32	64	128	256	512	1024
128	[0.	0.	0.	0.	0.89	0.92	0.96	0.99]
64	[0.	0.	0.	0.87	0.92	0.94	0.98	0.99]
32	[0.	0.	0.87	0.86	0.95	0.96	0.99	1.]
16	[0.	0.83	0.87	0.92	0.96	0.97	1.	1.]
8	[0.78	0.85	0.89	0.93	0.96	0.99	0.99	0.99]
4	[0.83	0.86	0.88	0.92	0.94	0.97	0.97	0.98]
2	[0.78	0.79	0.83	0.86	0.89	0.92	0.92	0.9]
1	[0.62	0.66	0.68	0.7	0.73	0.76	0.74	0.75]

FIG. 5: Table of normalized convergences for neighborhood size (y) versus swarm size (x). There is a slight advantage gained by choosing a reasonable neighborhood size.

4. *Dimensional Difficulty*

We varied the number of dimensions of the test function, expecting that the number of local optima exponentially depends on the number of dimensions. To avoid the problem of not knowing the true global maximum of our test functions, we invented an ad-hoc metric. For each test, we ran the PSO algorithm for 40 trials with the same parameters. We then found the maximum amongst

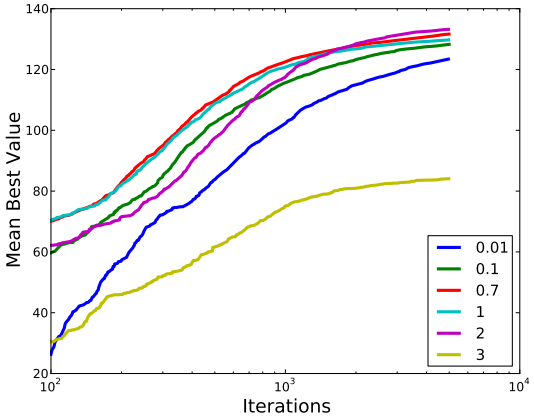

FIG. 6: We vary both of the spring constants in PSO. There is hardly any difference for values close to the standard input.

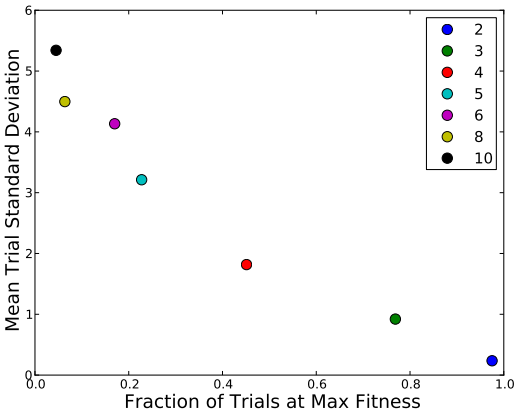

FIG. 7: 2d scatter plot, varying dimensionality of the landscape. We see the quick failure of PSO to find the global maximum in higher dimensions.

each of the 50 maxima. We then counted the number of maxima in the set of 50 that were within a small distance and a small fitness of the greatest maximum. For each dimension, we repeated this test 40 times, generating a new landscape for each test, and sweeping over several dimensions. The results are plotted across the x axis of Fig. 7. The y axis of this figure plots the mean standard deviation for the trials of each dimension. We see that almost every trial of the two dimensional function agreed on the maxima, thus it is overwhelmingly likely that the true global maxima was found. When more dimensions are added, the agreement percentage quickly drops off. With 9 dimensions, the highest maxima was only converged upon in one trial. This makes it exceedingly likely that the true global maximum was not found in any trial. The noise in the scatter plot is reasonable considering that at least 10^{14} floating point operations were conducted to acquire this data.

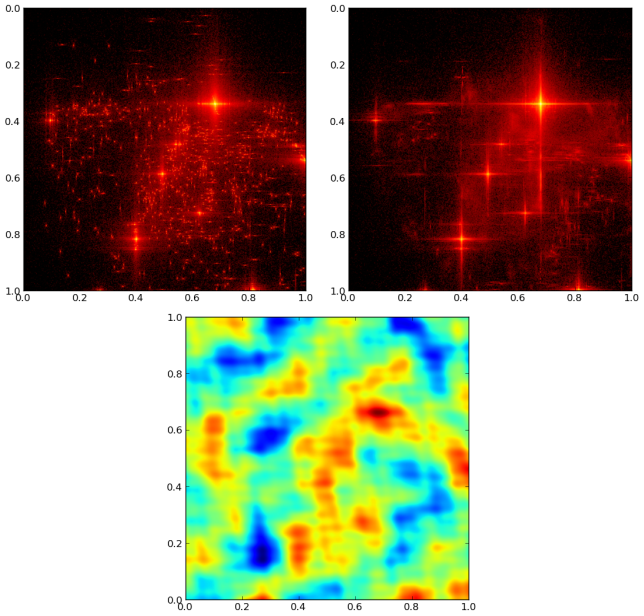

FIG. 8: Heat Maps of particle positions. The intensity of the top plots correspond to number of visits by a particle. The landscape is the bottom image. On the left, we have standard PSO. On the right, we have the Local Best Relocation tweak, which reduces the amount of time that particles spend at weak local maxima. The star effect is due to a basis vector bias in PSO.

5. *Dimensional Collapse*

We observed a position selection bias in the Naive PSO algorithm as described in the literature. It is often the case that a particle's local best and neighborhood best concur, at say position **b**. In this case, the particle acceleration is taken in each dimension as a random factor times the distance in that dimension from the particle:

$$a_i = k_1 U[0,1](b_i - x_i) + k_2 U[0,1](b_i - x_i) \qquad (5)$$

This causes a bias towards planes $\hat{\mathbf{i}} \cdot (\mathbf{x} - \mathbf{b}) = 0$. Thus, the algorithm is not basis independent. This phenomenon is visible in simulation heat maps as a star like effects surrounding the maxima (Fig. 8). One technique for avoiding this problem is to randomly perturb each velocity vector at every update.[6] However, upon attempting this, we saw a regression in convergence performance, and could not find a reasonable method to remedy this problem. A further problem plagues naive PSO in high dimensional problems. Even in a few dimensions, the particles tend to fall into sub-planes of the solution space as part of the convergence process.[6] However, analysis of this effect was challenging.

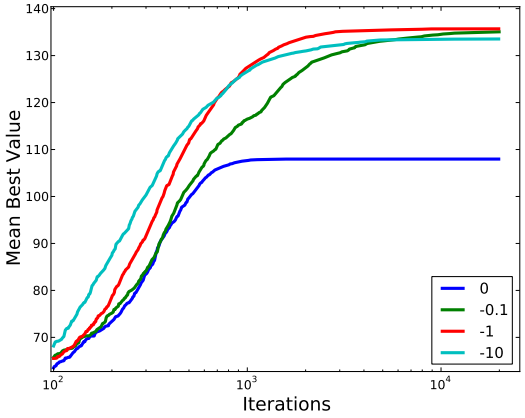

FIG. 9: Mean Fitness as a function of the local best reduction parameter. Using the parameter results in a significantly greater fitness.

6. *Local Best Reduction*

We found a tweak that substantially improves the performance of the naive PSO algorithm that we did not encounter in the literature. We are not aware of any existing mention of this technique in the literature. At every update, the local best value for each particle that does not post a better local best is decremented by some amount. This has the effect of moving the local best of some particles to a lower position. However, this tends to be beneficial to the swarm in avoiding early convergence. We observe the visual difference of this optimization in Fig. 8. We also found that this technique tended to do better than mutation techniques that randomly reinitialize some aspect of the swarm data.

Our technique was inspired by the greedy regression step in the successful WalkSAT CSP algorithm. The relocation of the local best is similarly greedy, because the decreasing threshold preferentially selects new local bests that are not much worse than the previous. Further work on this tweak could yield a formulation of PSO which no longer relies on the rigid definitions of local and neighborhood bests of standard PSO, but instead keeps some set of attraction points for the swarm determined determined by a more complex optimization algorithm.

IV. EVOLISA APPLICATION

A. Motivation

Computer vision is an important problem that is particularly salient to the study of artificial intelligence, as well as industries of security and engineering. PSO has been successfully applied to several difficult and important problems in computer vision, and has shown significant improvement for problems such as object recogni-

tion, object detection, and face recognition, where exhaustive search is computationally consuming. [7] [8] [9] Given PSO's success in similar problem spaces, we chose to apply PSO to the Evolisa problem. Evolisa algorithm is an interesting and novel approach to image matching, where a reference image is matched by a series of polygons. The polygonal representation of an image is useful in lowering the dimensionality for image comparison– where a high definition image might have millions of pixels, any image can be discretized using Evolisa into several hundred polygons. Even with the multiple dimensions of each polygon (vertexes and RGBA values), the dimensionality is still only in order of thousands. This can have nontrivial practical applications in computer vision problems of image processing where oftentimes pixel to pixel comparisons are particularly computationally heavy. Furthermore, output from Evolisa is resolution independent, which is a nice property for a compressed image, and allows artistically pleasing scalable versions of rasterized images.

B. Completeness

Many optimization problems have complex and high dimensional landscapes. For instance, in our implementation of the EvoLisa image matching algorithm, matching just 100 triangles with 10 dimensions each (6 coordinates and RGBA channels) constitutes a 1000 dimensional function. If the expected number of local maximum of a random 1-d slice of the landscape was just two, then the expected number of local maximum in the entire landscape could be as high as 2^{1000}, which is a completely intractable number. Completeness or even probability of reaching the global maximum on such functions is not a reasonable goal for a search algorithm. Instead, the metric of interest is often the efficiency of the search algorithm. This metric can be judged by investigating search algorithms that converge within the acceptable range of fitness most quickly.

C. PSO and Evolisa

The original Evolisa algorithm introduced by Alsing is described as a genetic programming algorithm. Genetic programming is a subset of genetic algorithms, utilizing the same genetic operators such as propagation, crossover, and mutation, but in which each individual in the population is a program.[10] Evolisa's population is essentially 2, a Parent and Child. In each generation, competition between the Parent and Child allows only the more fit individual to survive. The fitness is calculated by a pixel to pixel comparison of the difference in RGBA values between the reference and the current state. Alsing claims that the algorithm is an example of genetic programming, as each iteration of the application clones and mutates an executable Abstract Syntax Tree. However, since the population can only sustain one individual, it is in essence a hill-climbing algorithm as there is no propagation or chance of crossover.

The first iteration of the PSO Evolisa implementation was a pure PSO algorithm over the entire search state of the problem. This implementation chose to use triangles instead of mutating n-sided polygons in order to minimize computational complexity. Because the goal was time-efficient convergence, the loss of dimensionality in polygonal sidedness was a compromise between accuracy and optimized performance. In this implementation, each particle represents a complete image state with s triangles. Each triangle is represented by 10 dimensions– each vertex has two dimensions, and there are Red, Blue, Green and Alpha color channels. We found that using PSO directly on the entire state space was computationally and algorithmically foreboding. Each fitness evaluation would require shading all s triangles.

D. Combined hill-climbing and PSO

To avoid the computationally infeasible search of PSO over the entire state space, we instead ran a PSO instance on a single triangle in each iteration. This way, the partial solution for $s-1$ triangles could be rendered just once. Thus, every particle only represents a 10 dimensional solution to the final triangle. To further optimize the calculation of the fitness function, this implementation evaluates the global fitness of the complete solution by only summing over pixels covered by the final triangle. Let O be the goal image, let P be the partial solution of $s-1$ triangles, and let T be the rasterization of the final triangle. A equivalent fitness function (minimizing image difference) is:

$$|O-(P+T)|-|O-P| \tag{6}$$

This expression is non-zero only on pixels covered by T, and if the sum is negative, then the triangle is an improvement. The shading speedup here is huge when dealing with triangles that span a small area compared to the total image.

The last inequality of the new global best to the best fitness ever seen only allows beneficial changes to the overall image fitness, essentially implementing a hill-climbing algorithm in a specific feature, the triangles, are optimized with PSO. In practice, the speedup offered by this strategy was huge, and the disadvantages were not realized, since PSO failed to converge on the full problem.

Program 2 The PSOlisa algorithm

```
intialize population of S triangles
while (global best fitness > desired fitness):
    pick a triangle T at random from S:
    run PSO(T), moving T to optimized state
    if new global best > old global best:
        revert T to previous state
```

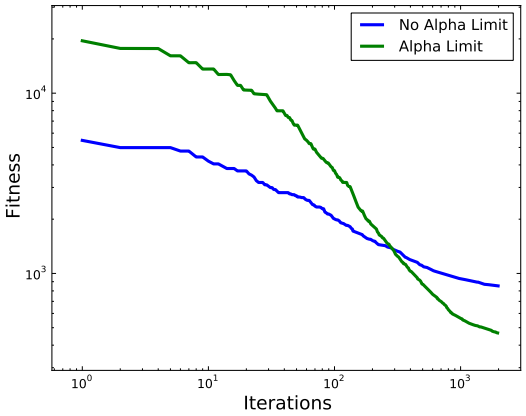

FIG. 10: Convergence to the Mona Lisa test image using no alpha limit (left), and using an alpha limit of 0.1 (right) both after 2000 iterations (500 triangles). In this plot, we are minimizing the fitness.

E. Limitations on Alpha value

One inefficiency that arose from optimizing one triangle at a time was that when few triangles were shaded, PSO favored triangles with alpha values closer to 1 as they initially showed a better fitness. Thus, in the first few iterations, PSO would favor partial solutions with non-overlapping triangles with high alpha values and bright color. However, as more triangles were added, this led to a large amount of noise, as the triangles with high alpha values created sharp lines of contrast in the image and particles that would easily prematurely converge on local optima, as seen in Fig. 10. One important optimization that was added to solve this problem was a limitation on the alpha values of triangles. Observing the fitness of an algorithm with an alpha limit, and an algorithm without an alpha limit, it is clear that the alpha limit not only visually prevents extraneous noise, but also shows a slightly slower but better convergence. The alpha limit encourages triangle overlap and results in a solution with far many more colors produce by different combinations of overlaid triangles. In Fig. 10 the 500 triangles are likely producing thousands of patches with distinct color. Through experimentation, we have found that the ideal alpha limit times the number of triangles is approximately 50.

FIG. 11: Convergence to the Mona Lisa test image using Roger Alsing's original Evolisa algorithm after 904314 iterations(left), and using PSOLisa algorithm with an alpha limit of 0.1 (right) both after 2000 iterations (500 triangles).

F. Performance

Mentioned in previous sections already are the consistent increased performance in convergence efficiency as we implemented the various parts of our PSOlisa algorithm. However, there is also an increase in convergence efficiency of our algorithm over traditional Evolisa. Outputs from both Alsing's Evolisa and PSOlisa shown in Fig. 11 converged after 90 seconds. Alsing's algorithm converged after 900,000 iterations while PSOlisa converged after 2000 iterations. However, each iteration of PSOlisa required more work. It is difficult to compare performance of the Alsing algorithm to PSOlisa, as empirical analysis of his implementation of mutation operators for adding and removing sides for his polygons is somewhat incompatible with the PSOlisa's fixed dimension algorithm. However, through subjective evaluation, PSOlisa produces a more subjectively visually pleasing and accurate Mona Lisa, and shows finer detail at a much higher resolution. In Evolisa, the presence of the polygonal shapes is much more apparent, as seen especially in the top left corner. The only drawback may be that PSOlisa shows a more noisiness in areas of flat color, such as in the background. However, the drawback of added noise in this case is overshadowed by the benefits of finer detail in features such as the face.

V. CONCLUSION

In this paper we presented empirical experimental results of several parametrizations of PSO over a stochastic fractal-like objective function. These parameter sweep experiments test the effect of parametrization of neighborhood size, dampening factor, spring constants and dimensionality on the robustness of PSO to find global optima and avoid premature convergence. Our experimental results show that efficient optimization of parameters can severely affect convergence performance. We further propose a novel optimization technique of local best relocation to prevent premature convergence that

surpassed other tweaks to PSO that are mentioned in the literature. Furthermore, this paper proposes an efficient combined hill-climbing and PSO algorithm to solve the Evolisa image matching problem. Image matching is a very computationally heavy problem, so an optimization that leads to the same fitness with fewer function evaluations is important. These results have potential impact in fields of image recognition and computer vision, and are aesthetically interesting.

The experimental findings in this paper are a promising beginning to understanding the full effects of parametrization in PSO. Because parameters in PSO are all interdependent, the experiments in this paper can be expanded in the future to examine parameter co-dependencies and interdependencies in a multi-dimensional parameter space. Our modification PSO to include the local best relocation caused us to rethink whether standard PSO is the most efficient parallel optimization algorithm for continuous spaces.

[1] J. Kennedy and R. C. Eberhart, "Particle Swarm Optimization," Proc. IEEE int'l conf. on neural networks **IV**, 1942 (1995).

[2] R. Alsing, "Genetic Programming: Evolution of Mona Lisa,", `http://rogeralsing.com/2008/12/07/genetic-programming-evolution-of-mona-lisa/` (2009).

[3] A. Klöckner, N. Pinto, Y. Lee, B. Catanzaro, P. Ivanov, and A. Fasih, "PyCUDA: GPU Run-Time Code Generation for High-Performance Computing," Cornell University Library (2009), 0911.3456, URL `http://arxiv.org/abs/0911.3456`.

[4] J. Li, B. Ren, and C. Wang, in *LSMS (1)* (2007), pp. 92–99.

[5] URL `http://www.pitbear.com/eng-ga4ts.php/`.

[6] K. U. Hatanaka, Korenaga, "Search Performance Improvement for PSO in High Dimensional Space," Particle Swarm Optimization (2009).

[7] H. A. Perlin, H. S. Lopes, and T. M. Centeno, "Particle Swarm Optimization for Object Recognition in Computer Vision," New Frontiers in Applied Artificial Intelligence **5027**, 11 (2008).

[8] A. W. Mohemmed, M. Zhang, and M. Johnston, "A PSO Based Adaboost Approach to Object Detection," Simulated Evolution and Learning **5361**, 81 (2008).

[9] R. M. Ramadan and R. F. Abdel, "Face Recognition Using Particle Swarm Optimization-Based Selected Features," International Journal of Signal Processing, Image Processing and Pattern Recognition **2.2** (2009).

[10] W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone, *Genetic Programming: An Introduction* (Morgan Kaufmann, 1997).