A Quick Analysis of Heuristic Optimization by Stochastic Genetic Algorithms and Particle Swarm

We investigate the optimization performance of both Genetic Algorithms (GAs) and Particle Swarm Optimization (PSO). Particularly we are interested in comparing the efficiency and accuracy of these techniques in optimizing a variety of well known test functions. The test functions used are outlined in Appendix A. We consider three classes of GAs:

- Matlab's Built in Traditional GA (MatlabGA),
- A traditional GA as outlined by Varadi in *http://cssanalytics.wordpress.com/* (GA)
- A simplistic Micro-GA (mGA) as outlined by Varadi in *http://cssanalytics.wordpress.com/*.

A Brief Review of Traditional GAs

Recall, like most evolutionary algorithms, a GA is a stochastic sampling method that repeatedly refines a population or set of candidate solutions. The method is based on the Darwinian notion of "Survival of the Fittest" in which only the *elite* solutions, (those that return the most optimal objective function values) are kept, while sub-optimal solutions are discarded.

A typical GA begins by randomly generating an initial population candidate solutions, called the parents. The initial population of each generation is then enlarged through production of a combination of *cross-over children* and/or *mutation children*. Cross-over children are derived by mixing and matching individual components of the parents, in what Varadi describes as Mating. Formally, at each component i of the child vector, the default crossover function randomly selects a value, x_i, from one of the two parents and assigns it to the i^{th} component of the cross-over child. Mutation children are created by randomly perturbing the parent solutions, often via a Gaussian distribution.

Next we evaluate the corresponding objective function value at each of the potential solution in the current population. Only a fraction of the individuals in the current population that provide the most optimal objective function value will continue onward to form the parent population of the next generation.

The mGA, on the other hand, is a simplified version of the traditional GA and involves a Tournament phase for determining which parent solutions are worth to mate. Those parents that are worthy to mate produce cross-over children, as described above and in *http://cssanalytics.wordpress.com/*.

For consistency amongst the varying GAs we have kept each population size and maximum number of generations fixed at 100 and $100d$, respectively, where d is the number of input dimensions to the test function. Convergence is established either by achieving the maximum number of generations or if the average of the best solution found over the most recent 50 generations is less than a tolerance, $tol = 10^{-6}$.

A Brief Review of PSO

PSO is a gradient free, stochastic optimization algorithm. Like GAs and mGAs, PSO begins by first generating an initial set of candidate solutions or particles, collectively called the swarm. At each iteration, individual particles of the swarm navigate the search-space based on each particles personal best position and the overall best position found by the swarm. Therefore, previous locations that minimize the objective function, f, act as attractors to which the swarm will migrate towards. The position of each particle, and the swarm alike, is defined in terms of its previous position, x, and the velocity, v, of the particle(or swarm). The velocity (or momentum factor) describes a particles tendency to move in a certain direction. The position and velocity are defined as:

$$x_{n+1} = x_n + v_n$$

and

$$v_{n+1} = v_n + r_1(x_p - x_n) + r_2(x_s - x_n)$$

where r_1 and r_2 are random numbers on the interval $(0, 1)$, x_p denotes the best previous position of that particle, and x_s denotes the best previous position of the swarm.

For optimization of each test function I have chosen to use a swarm size of 20. For a fair comparison with the GAs, the stopping criteria for PSO is that the average of the best solution found over the most recent 50 iterations is less than a tolerance, $tol = 10^{-6}$.

Lastly, note that since both the GA and PSO are stochastic optimization routines, the results presented below have each been averaged over 100 simulations.

Numerical Results

We analyze three quantities:

- **Accuracy:** Average Percent Error $= \frac{|f^* - f_{opt}|}{|f_{opt}|} \times 100\%$, where f_{opt} is the true, known, global optimum value.
- **Efficiency:** Average number of Function Evaluations (FEs) required for convergence.
- **General Performance Assessment:** GPA= FEs $\times (\%Err)^2$, in which accuracy holds twice the weight as efficiency and smaller GPA values indicate optimal performance.

Simulation results for each test function can be found in Table 1. From Table 1 we can draw the following conclusions:

- In general MatlabGA outperforms both my own traditional GA and mGA, with the exception of optimizing the 2-D Hump Function and 6-D Hartmann function.
  - This is not surprising given the level of sophistication of Matlab's built in optimization algorithms as compared to the 1-day coding blitz of my GAs.
- The mGA is indeed the most efficient of the GAs, as measured by the number of FEs required to establish convergence.

  - Of course improved efficiency comes at the cost of decreased accuracy - ultimately the user's optimization goals will play a huge factor in algorithm selection.

- As expected, PSO outperforms all GAs for objective functions that are strictly convex, with minimal noise and a single global optimum.
  - As Varadi suggests, PSO (or a hybrid PSO method) may be the preferred technique for several portfolio optimization problems.
  - Objective surfaces such as the Schwefel Function and the Rastrigin Function are noisy and multi-modal, containing several local optima. For global optimization of both these functions, MatlabGA proves to outperform PSO.

Algorithm	**Hump (2-D)**			**Goldstein-Price(2-D)**		
	%*Err.*	FEs	GPA	%*Err.*	FEs	GPA
MatlabGA	0.0317	5200	5.2	5.84e-5	5200	1.8e-5
GA	0.0316	5401	7.0	0.104	9279	100
mGA	0.0358	2861	3.7	200	2974	1.2e8
PSO	0.0316	2040	<u>2.0</u>	2.65e-9	2040	<u>1.4e-14</u>
	Schwefel (5-D)			**Hartmann (6-D)**		
	%*Err.*	FEs	GPA	%*Err.*	FEs	GPA
MatlabGA	38.0	5801	<u>8.4e6</u>	1.87	5278	1.7e4
GA	96.4	9890	9.2e7	0.925	9930	8.5e3
mGA	344	3505	4.2e8	6.155	4130	1.6e5
PSO	367	5639	7.6e8	1.01	6268	<u>6.4e3</u>
	Rosenbrock (10-D)			**Rastrigin (10-D)**		
	%*Err.*	FEs	GPA	%*Err.*	FEs	GPA
MatlabGA	45.2	9677	2.0e7	3.57	5964	<u>7.6e4</u>
GA	3300	9782	1.1e11	13.2	9848	1.7e6
mGA	2396	4692	2.7e10	33.4	3463	3.8e6
PSO	0.729	46540	<u>2.5e4</u>	19.8	10082	4.0e6

Table 1: Performance comparison of different GA and PSO optimization techniques on six test functions. Underlined values in the GPA column indicate the best GPA value.

A Test Functions

Test Function (d)	**Formula y=f(x)**
Hump (2)	$y = 1.0316285 + 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$
Goldstein-Price (2)	$y =$ $[1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)]\times$ $[30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)]$
Schwefel (5)	$y = 2094.9 - \sum_{i=1}^{5} x_i \sin\left(\sqrt{\lvert x_i \rvert}\right)$
Hartmann (6)	$y = -\sum_{i=1}^{6} \alpha \cdot \exp[-\sum_{j=1}^{6} B_{ij}(x_j - Q_{ij})^2]$ $\alpha = [1, 1.2, 3, 3.2], \quad B = \begin{bmatrix} 10 & 3 & 17 & 3.5 & 1.7 & 8 \\ 0.02 & 10 & 17 & 0.1 & 8. & 14 \\ 3 & 3.5 & 1.7 & 10 & 17 & 8 \\ 17 & 8 & 0.05 & 10 & 0.1 & 14 \end{bmatrix},$ $Q = \begin{bmatrix} 0.1312 & 0.1696 & 0.5569 & 0.0124 & 0.8283 & 0.588 \\ 0.2329 & 0.4135 & 0.8307 & 0.3736 & 0.1004 & 0.9991 \\ 0.2348 & 0.1451 & 0.3522 & 0.2883 & 0.3047 & 0.6650 \\ 0.4047 & 0.8828 & 0.8732 & 0.5743 & 0.1091 & 0.0381 \end{bmatrix}$
Rastrigin (10)	$y = 10n + \sum_{i=1}^{10}(x_i^2 - 10\cos(2\pi x_i))$
Rosenbrock (10)	$y = \sum_{i=1}^{9}[100(x_i^2 - x_{i+1})^2 + (x_i - 1)^2]$

Table 2: Test functions and corresponding formula used for evaluating the performance of the GA and PSO optimization algorithms.

For full test function details see:
http://www-optima.amp.i.kyoto-u.ac.jp/member/student/hedar/Hedar_files/TestGO.htm