AIA[illegible]1897

A COMPARISON OF PARTICLE SWARM OPTIMIZATION AND THE GENETIC ALGORITHM

Rania Hassan* Babak Cohanim† Olivier de Weck‡

Massachusetts Institute of Technology, Cambridge, MA, 02139

Gerhard Venter§
Vanderplaats Research and Development, Inc., Colorado Springs, CO, 80906

Particle Swarm Optimization (PSO) is a relatively recent heuristic search method whose mechanics are inspired by the swarming or collaborative behavior of biological populations. PSO is similar to the Genetic Algorithm (GA) in the sense that these two evolutionary heuristics are population-based search methods. In other words, PSO and the GA move from a set of points (population) to another set of points in a single iteration with likely improvement using a combination of deterministic and probabilistic rules. The GA and its many versions have been popular in academia and the industry mainly because of its intuitiveness, ease of implementation, and the ability to effectively solve highly nonlinear, mixed integer optimization problems that are typical of complex engineering systems. The drawback of the GA is its expensive computational cost. This paper attempts to examine the claim that PSO has the same effectiveness (finding the true global optimal solution) as the GA but with significantly better computational efficiency (less function evaluations) by implementing statistical analysis and formal hypothesis testing. The performance comparison of the GA and PSO is implemented using a set of benchmark test problems as well as two space systems design optimization problems, namely, telescope array configuration and spacecraft reliability-based design.

Nomenclature

c_1	=	self confidence factor
c_2	=	swarm confidence factor
f	=	fitness function
g	=	constraint function
$H_\circ$	=	null hypothesis
H_a	=	alternative hypothesis
n	=	sample size
N_{feval}	=	number of function evaluations
$\mathbf{p}_k^g$	=	position of the particle with best global fitness at current move k
$\mathbf{p}^i$	=	best position of particle i in current and all previous moves
r	=	penalty multiplier

* Postdoctoral Associate, Aeronautics & Astronautics and Engineering Systems, rhassan@mit.edu, AIAA Member.
† Formerly, Graduate Student, Center for Space Research, Aeronautics & Astronautics. Currently, Systems Engineer, The Jet Propulsion Laboratory, bec@alum.mit.edu, AIAA Member.
‡ Assistant Professor, Aeronautics & Astronautics and Engineering Systems, deweck@mit.edu, AIAA Senior Member.
§ Senior R&D Engineer, gventer@vrand.com, AIAA Member.

$rand$ = uniformly distributed rand variable between 0 and 1
Q_{sol} = solution quality
$s(x)$ = estimate of the standard deviation of a population from a sample
$s(\bar{x})$ = the standard deviation of a sample
t = t-statistic
$\mathbf{v}_k^i$ = position of particle i in the design space at time k
w = inertia factor
$\bar{x}$ = sample mean
$\mathbf{x}$ = design vector
$\mathbf{x}_k^i$ = position of particle i in the design space at time k
α = significance level of a test
β = power of the test
μ = mean
Δt = time increment
ϕ = objective function

Introduction

Particle Swarm Optimization (PSO) was invented by Kennedy and Eberhart[1] in the mid 1990s while attempting to simulate the choreographed, graceful motion of swarms of birds as part of a socio-cognitive study investigating the notion of "collective intelligence" in biological populations. In PSO, a set of randomly generated solutions (initial swarm) propagates in the design space towards the optimal solution over a number of iterations (moves) based on large amount of information about the design space that is assimilated and shared by all members of the swarm. PSO is inspired by the ability of flocks of birds, schools of fish, and herds of animals to adapt to their environment, find rich sources of food, and avoid predators by implementing an "information sharing" approach, hence, developing an evolutionary advantage. References 1 and 2 describe a complete chronicle of the development of the PSO algorithm from merely a motion simulator to a heuristic optimization approach.

The Genetic Algorithm (GA) was introduced in the mid 1970s by John Holland and his colleagues and students at the University of Michigan.[3] The GA is inspired by the principles of genetics and evolution, and mimics the reproduction behavior observed in biological populations. The GA employs the principal of "survival of the fittest" in its search process to select and generate individuals (design solutions) that are adapted to their environment (design objectives/constraints). Therefore, over a number of generations (iterations), desirable traits (design characteristics) will evolve and remain in the genome composition of the population (set of design solutions generated each iteration) over traits with weaker undesirable characteristics. The GA is well suited to and has been extensively applied to solve complex design optimization problems because it can handle both discrete and continuous variables, and nonlinear objective and constraint functions without requiring gradient information.

The major objective of this paper is to compare the computational effectiveness and efficiency of the GA and PSO using a formal hypothesis testing approach. The motivation is to validate or refute the widely speculated hypothesis that PSO has the same effectiveness as the GA (same rate of success in finding true global optimal solutions) but with better computational efficiency. The results of this test could prove to be significant for the future development of PSO.

The remainder of this paper is organized in five major sections. First, the PSO and GA versions that are implemented in this comparative study are summarized and the hypothesis test procedure is formulated. Second, three well-known benchmark problems that are used to compare the performance of the GA and PSO are presented. Third, two space systems optimization problems that are used to test the performance of both algorithms with respect to real life applications are presented. Results and conclusions are presented in the last two sections.

PSO versus GA

Particle Swarm Optimization

In this study, the basic PSO algorithm that is described in Reference 4 is implemented. The basic algorithm is first described, followed by a discussion on side and functional constraint handling, and finally, a discrete version of the algorithm is presented. It should be noted that while the GA is inherently discrete, i.e. it encodes the design

variables into bits of 0's and 1's, therefore it easily handles discrete design variables, PSO is inherently continuous and must be modified to handle discrete design variables.

The basic PSO algorithm consists of three steps, namely, generating particles' positions and velocities, velocity update, and finally, position update. Here, a particle refers to a point in the design space that changes its position from one move (iteration) to another based on velocity updates. First, the positions, $\mathbf{x}_k^i$, and velocities, $\mathbf{v}_k^i$, of the initial swarm of particles are randomly generated using upper and lower bounds on the design variables values, $\mathbf{x}_{\min}$ and $\mathbf{x}_{\max}$, as expressed in Equations 1 and 2. The positions and velocities are given in a vector format with the superscript and subscript denoting the i^{th} particle at time k. In Equations 1 and 2, *rand* is a uniformly distributed random variable that can take any value between 0 and 1. This initialization process allows the swarm particles to be randomly distributed across the design space.

$$\mathbf{x}_0^i = \mathbf{x}_{\min} + rand\left(\mathbf{x}_{\max} - \mathbf{x}_{\min}\right) \tag{1}$$

$$\mathbf{v}_0^i = \frac{\mathbf{x}_{\min} + rand\left(\mathbf{x}_{\max} - \mathbf{x}_{\min}\right)}{\Delta t} = \frac{\text{position}}{\text{time}} \tag{2}$$

The second step is to update the velocities of all particles at time $k+1$ using the particles objective or fitness values which are functions of the particles current positions in the design space at time k. The fitness function value of a particle determines which particle has the best global value in the current swarm, $\mathbf{p}_k^g$, and also determines the best position of each particle over time, $\mathbf{p}^i$, i.e. in current and all previous moves. The velocity update formula uses these two pieces of information for each particle in the swarm along with the effect of current motion, $\mathbf{v}_k^i$, to provide a search direction, $\mathbf{v}_{k+1}^i$, for the next iteration. The velocity update formula includes some random parameters, represented by the uniformly distributed variables, *rand*, to ensure good coverage of the design space and avoid entrapment in local optima. The three values that effect the new search direction, namely, current motion, particle own memory, and swarm influence, are incorporated via a summation approach as shown in Equation 3 with three weight factors, namely, inertia factor, w, self confidence factor, c_1, and swarm confidence factor, c_2, respectively.

velocity of particle i at time k+1

$$\mathbf{v}_{k+1}^i = w\,\mathbf{v}_k^i + c_1\,rand\,\frac{\left(\mathbf{p}^i - \mathbf{x}_k^i\right)}{\Delta t} + c_2\,rand\,\frac{\left(\mathbf{p}_k^g - \mathbf{x}_k^i\right)}{\Delta t} \tag{3}$$

current motion

particle memory influence

swarm influence

inertia factor range: 0.4 to 1.4

self confidence range: 1.5 to 2

swarm confidence range: 2 to 2.5

The original PSO algorithm[1] uses the values of 1, 2 and 2 for w, c_1, and c_2 respectively, and suggests upper and lower bounds on these values as shown in Equation 3 above. However, the research presented in this paper found out that setting the three weight factors w, c_1, and c_2 at 0.5, 1.5, and 1.5 respectively provides the best convergence rate for all test problems considered. Other combinations of values usually lead to much slower convergence or sometimes non-convergence at all. The tuning of the PSO algorithm weight factors is a topic that warrants proper investigation but is outside the scope of this work. For all the problems investigated in this work, the weight factors use the values of 0.5, 1.5 and 1.5 for w, c_1, and c_2 respectively.

Position update is the last step in an iteration. The Position of each particle is updated using its velocity vector as shown in Equation 4 and depicted in Figure 1. A constant unit time step, i.e. $\Delta t = 1$, is taken throughout the PSO algorithm in Equations 2, 3 and 4.

$$\mathbf{x}_{k+1}^i = \mathbf{x}_k^i + \mathbf{v}_{k+1}^i\,\Delta t \tag{4}$$

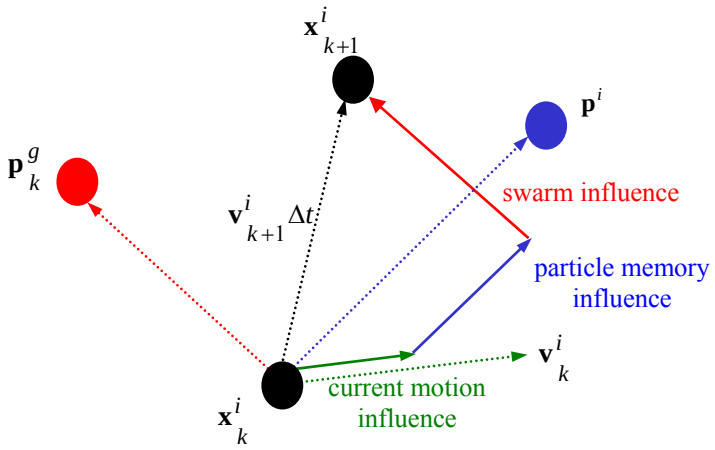

Figure 1. Depiction of the velocity and position updates in Particle Swarm Optimization.

The three steps of velocity update, position update, and fitness calculations are repeated until a desired convergence criterion is met. In the PSO algorithm implemented in this study, the stopping criteria is that the maximum change in best fitness should be smaller than an absolute tolerance for a specified number of moves, S, as shown in Equation 5. In this work, S is specified as ten moves and ε is specified as 10^{-5} for all test problems.

$$\left| f\left(\mathbf{p}^g_k\right) - f\left(\mathbf{p}^g_{k-q}\right)\right| \le \varepsilon \quad q = 1,2,..S \tag{5}$$

Unlike the GA with its binary encoding, in PSO, the design variables can take any values, even outside their side constraints, based on their current position in the design space and the calculated velocity vector. This means that the design variables can go outside their lower or upper limits, $\mathbf{x}_{\min}$ or $\mathbf{x}_{\max}$, which usually happens when the velocity vector grows very rapidly; This phenomenon can lead to divergence. To avoid this problem, in this study, whenever the design variables violate their upper or lower design bounds, they are artificially brought back to their nearest side constraint. This approach of handling side constraints is recommended by Reference 4 and is believed to avoid velocity "explosion".[2] Functional constraints on the other hand are handled in the same way they are handled in the GA. In this comparative study, functional constraints are handled using a linear exterior penalty function approach as shown in Equation 6.

$$f(\mathbf{x}) = \phi(\mathbf{x}) + \sum_{i=1}^{N_{con}} r_i \max\left[0, g_i(\mathbf{x})\right] \tag{6}$$

In addition to applying penalty functions to handle designs with violated functional constraints, in PSO, it is recommended that the velocity vector of a particle with violated constraints be reset to zero in the velocity update formula as shown in Equation 7.[4] This is because if a particle is infeasible, there is a big chance that the last search direction (velocity) was not feasible. This approach is implemented in this study.

$$\mathbf{v}^i_{k+1} = c_1\, rand\, \frac{\left(\mathbf{p}^i - \mathbf{x}^i_k\right)}{\Delta t} + c_2\, rand\, \frac{\left(\mathbf{p}^g_k - \mathbf{x}^i_k\right)}{\Delta t} \tag{7}$$

In many design applications, especially complex systems, the optimization problem statement usually includes discrete design variables, such as technology choices. Reference 4 suggests a simple but effective way to implement discrete design variables with PSO, that is to round particle position coordinates (design variable values) to the nearest integers when the design variables being investigated are discrete. This is the strategy that is implemented in this study.

There has been no recommendation in the literature regarding swarm size in PSO. Most researchers use a swarm size of 10 to 50 but there is no well established guideline.[2] For the purpose of comparing PSO and GA, the swarm

size that will be used for the PSO runs in all test problems will be the same as the population size in their equivalent GA runs and is fixed at 40 particles in the PSO swarm and 40 chromosomes in GA population.

The Genetic Algorithm

The literature includes many versions of the Genetic Algorithm (GA). In this study, a basic binary encoded GA[5] with tournament selection, uniform crossover and low probability mutation rate is employed to solve the benchmark problems and the space systems design problems. The GA represents the design variables of each individual design with binary strings of 0's and 1's that are referred to as chromosomes. It is important to note that the GA works with a coding for the design parameters that allows for a combination of discrete and continuous parameters in one problem statement. This encoding feature also forces the design variables to only take values that are within their upper and lower bounds, i.e., no solutions will ever violate the side constraints and infeasibility can only occur because of violation of functional constraints.

Like PSO, the GA begins its search from a randomly generated population of designs that evolve over successive generations (iterations), eliminating the need for a user-supplied starting point. To perform its optimization-like process, the GA employs three operators to propagate its population from one generation to another. The first operator is the "Selection" operator that mimics the principal of "Survival of the Fittest". The second operator is the "Crossover" operator, which mimics mating in biological populations. The crossover operator propagates features of good surviving designs from the current population into the future population, which will have better fitness value on average. The last operator is "Mutation", which promotes diversity in population characteristics. The mutation operator allows for global search of the design space and prevents the algorithm from getting trapped in local minima. Many references explain the details of the GA mechanics of evolution; see, for example, Reference 3. The specifics of the GA version implemented in this study are partially based on empirical studies developed in Reference 5, which suggests the combination of tournament selection with a 50% uniform crossover probability. For the comparison with PSO, the population size is kept constant at 40 chromosomes for all problems under consideration. A small, fixed mutation of 0.5% is implemented.

The basic GA version implemented in this study uses a linear exterior penalty function approach to handle functional constraints similar to PSO as shown in Equation 6 above. The GA also uses the same convergence criteria as PSO as shown in Equation 5.

Comparison Metrics and Hypothesis Testing

The objective of this research is to statistically compare the performance of the two heuristic search methods, PSO and the GA, using a representative set of test problems that are of diverse properties. The *t*-test (hypothesis testing) is used to assess and compare the effectiveness and efficiency of both search algorithms. In hypothesis testing, a null hypothesis, $H_\circ$ will be correctly accepted with a significance (or confidence) level $(1-\alpha)$ and falsely rejected with a type I error probability α. If the null hypothesis is false, it will be correctly rejected with a power of the test $(1-\beta)$ and will be falsely accepted with a type II error probability β. The decision options summary is presented in Table 1. H_a corresponds to an alternative hypothesis that is complimentary to $H_\circ$.

Table 1. Possible decision outcomes in hypothesis testing (adapted from Reference 6).

Action	The true situation may be	
	$H_\circ$ is true	$H_\circ$ is false
accept $H_\circ$, reject H_a	$(1-\alpha)$ significance level	β [type II error]
reject $H_\circ$, accept H_a	α [type I error]	$(1-\beta)$ power of test
Sum	1	1

The *t*-test deals with the estimation of a true value from a sample and the establishing of confidence ranges within which the true value can said to lie with a certain probability $(1-\alpha)$. In hypothesis testing, increasing the sample size, *n*, decreases type I and II error probabilities, α and β, based on the *t*-distribution. When *n* is very large, the *t*-distribution approaches the normal distribution.[7]

Hypothesis testing involves five steps. First, the null hypothesis and the alternative hypothesis are defined. The hypotheses could be two sided or single sided. For example, $H_\circ : \mu = m$ (with $H_a : \mu \neq m$) is a two sided hypothesis testing whether an unknown population mean, μ, is equal to the value *m* or not. On the other hand, $H_\circ : \mu > m$ (with $H_a : \mu \leq m$) is a single sided hypothesis. The second step is to decide on desired values for α, β, and n and to find out the corresponding $t_{critical}$ value for the given α, β, and n parameters for the specific type of test under consideration (single or two sided). The $t_{critical}$ value could be obtained from *t*-distribution tables that are available in many basic statistics textbook such as Reference 7. The third step is to evaluate *n* random samples and evaluate the sample mean, $\bar{x}$, an estimate of the standard deviation of a population from the sample, $s(x)$, and finally, the sample mean, $s(\bar{x})$, as shown in Equations 8 through 10.

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n} \tag{8}$$

$$s(x) = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2} \tag{9}$$

$$s(\bar{x}) = \frac{s(x)}{\sqrt{n}} \tag{10}$$

The fourth step in the hypothesis testing process is to calculate the *t*-value of the null hypothesis as shown in Equation 11. The formula for calculating the *t*-value differs according to the hypothesis being tested. The form shown in Equation 11 is equivalent to the two sided test $H_\circ : \mu = m$ (with $H_a : \mu \neq m$). Reference 7 lists the most important t-value calculation formulas.

$$t = \frac{|\bar{x} - m|}{s(\bar{x})} \tag{11}$$

The final step in the hypothesis testing process is to compare the calculated *t*-value to the tabulated $t_{critical}$ value. If $t_{calculated} \leq t_{critical}$, the null hypothesis is then accepted with $(1-\alpha)$ confidence level.

In this study, two hypotheses are tested. The first test is related to the effectiveness (finding the true global optimum) of the algorithms and the second is related to the efficiency (computational cost) of the algorithms. The first test is the high effectiveness test. Effectiveness is defined as the ability of the algorithm to repeatedly find the known global solution or arrive at sufficiently close solutions when the algorithm is started from many random different points in the design space. In other words, effectiveness is defined as a high probability of finding a high quality solution. Here, the quality of a solution is measured by how close the solution is to the known global solution as shown in Equation (12).

$$Q_{sol} = \left| \frac{\text{solution - known solution}}{\text{known solution}} \right| \% \tag{12}$$

The solution quality metric described in Equation 12 could then be used to synthesize a meaningful hypothesis to test the effectiveness of the search algorithms, PSO and the GA as shown below. It must be notes that the effectiveness test must be carried out for each of the algorithms separately. In other words, this test measures the effectiveness of each algorithm with respect to known solutions for the test problem rather than comparing the effectiveness of the two algorithms to each others. This effectiveness test will be carried out for each algorithm with respect to each of the test problems separately.

Effectiveness Test

Objective to test whether H_a : $\mu_{Q_{sol}} > 99\%$

$H_\circ$: $\mu_{Q_{sol}} \leq 99\%$

$$t = \frac{\overline{Q}_{sol} - 99\%}{s(\overline{Q}_{sol})}$$

taking $\alpha = 1\%$, $\beta = 1\%$, and $n = 10$

This is a one sided test of significance of a mean (table 6.10, Reference 7) $\rightarrow$ $t_{critical} = 2.0$

The second hypothesis that is tested in this study, which is the important test from the authors' point of view, is the computational efficiency test. This test directly compares the computational effort required by PSO and the GA to solve each of the test problems. The objective is to support or refute the widely speculated claim that PSO is significantly more efficient than the GA. This direct comparison requires a *t*-test called "comparison of two means". For a test with the null hypothesis $H_\circ$: $\mu_1 \leq \mu_2$, the *t*-value can be calculated as shown in Equation (13).

$$t = \frac{\bar{x}_1 - \bar{x}_2}{\bar{s}(x)\sqrt{1/n_1 + 1/n_2}} \tag{13}$$

where $$\bar{s}(x) = \sqrt{\frac{(n_1 - 1)s^2(x_1) + (n_2 - 1)s^2(x_2)}{n_1 + n_2 - 2}} \tag{14}$$

For the computational efficiency test, the metric that is implemented is the number of function evaluations, N_{feval} , the algorithm performed until the convergence criteria (described in Equation 5) is met. The lower N_{feval} is, the more efficient the algorithm. The efficiency test is summarized as follows.

Efficiency Test

Objective to test whether H_a : PSO $\mu_{N_{eval}}$ < GA $\mu_{N_{eval}}$

$H_\circ$: PSO $\mu_{N_{eval}} \geq$ GA $\mu_{N_{eval}}$

$$t = \frac{\text{GA } \bar{\mu}_{N_{feval}} - \text{PSO } \bar{\mu}_{N_{feval}}}{\bar{s}(x)\sqrt{1/n_{\text{GA}} + 1/n_{\text{PSO}}}} \text{ where } \bar{s}(x) = \sqrt{\frac{(n_{\text{GA}} - 1)s_{\text{GA}}^2 + (n_{\text{PSO}} - 1)s_{\text{PSO}}^2}{n_{\text{GA}} + n_{\text{PSO}} - 2}}$$

taking $\alpha = 1\%$, $\beta = 1\%$, and $n_{\text{PSO}} = n_{\text{GA}} = 10$

This is a one sided test of significance of comparison of two means (table 6.11, Reference 7) $\rightarrow$ $t_{critical} = 2.5$

Benchmark Test Problems

In this section, a set of well-known optimization benchmark test problems are presented. These problems will be solved by both PSO and the GA, ten times each, to carry out the hypothesis testing procedure as discussed in the previous section. This set of problems includes two simple, unconstrained continuous functions, namely, the Banana (Rosenbrock) function and the Eggcrate function; each has only two design variables. A more complex and highly constrained benchmark problem, namely, Golinski's Speed Reducer, is also presented; it has seven design variables and 11 functional constraints. The solutions for these test problems are known and will be used to evaluate the effectiveness of the two algorithms, PSO and the GA.

The Banana (Rosenbrock) Function

This function is known as the "banana function" because of its shape; it is described mathematically in Equation 15. In this problem, there are two design variables with lower and upper limits of [-5 , 5]. The Rosenbrock function has a known global minimum at [1 , 1] with an optimal function value of zero.

$$\text{Minimize} \quad f(\mathbf{x}) = 100\left(x_2 - x_1^2\right)^2 + (1 - x_1)^2 \tag{15}$$

The Eggcrate Function

This function is described mathematically in Equation 16. In this problem, there are two design variables with lower and upper bounds of [-2π , 2π]. The Eggcrate function has a known global minimum at [0 , 0] with an optimal function value of zero.

$$\text{Minimize} \quad f(\mathbf{x}) = x_1^2 + x_2^2 + 25\left(\sin^2 x_1 + \sin^2 x_2\right) \tag{16}$$

Golinski's Speed Reducer

This problem represents the design of a simple gearbox such as might be used in a light airplane between the engine and propeller to allow each to rotate at its most efficient speed. The gearbox is depicted in Figure 2 and its seven design variables are labeled. The objective is to minimize the speed reducer's weight while satisfying the 11 constraints imposed by gear and shaft design practices. Full problem description can be found in Reference 8. A known feasible solution obtained by a sequential quadratic programming (SQP) approach (MATLAB's fmincon) is a 2994.34 units mass gearbox with the following optimal values for the seven design variables: [3.5000 0.7000 17.0000 7.3000 7.7153 3.3502 5.2867]. This is a feasible solution with four active constraints. It is important to note that the published solution in Reference 8 is not a feasible solution as is widely believed because it slightly violates the 5th, 6th and 11th functional constraints.[9]

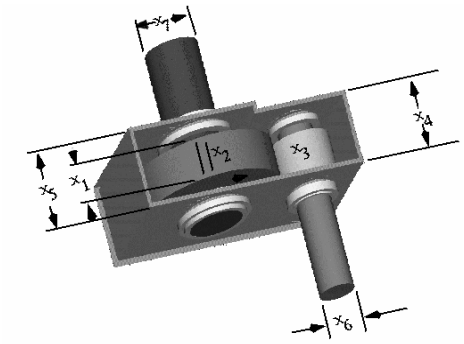

Figure 2. Golinski's Speed Reducer.

Space Systems Problems

This section presents two space system design problems; the first problem has a scientific application while the second focuses on commercial services. The two problems presented in this section, namely, ground-based radio telescope array design, and communication satellite design, are used as test problems to compare the performance of the two algorithms, PSO and GA, with respect to real life engineering problems.

Telescope Array Configuration

Ground-based radio telescope arrays use several distributed small telescopes to achieve the resolution of a costly single large telescope by correlating data to create a synthesized beam. A measure of performance for correlation arrays is the degree of coverage in the instantaneous uv plane. The uv plane is defined by Equation 17 where x and y are the ground positions of the stations measured in a convenient set of units such as kilometers.

$$u_{ij} = x_j - x_i \quad , \quad v_{ij} = y_j - y_i \qquad i, j = 1, 2, \ldots N_{stations} \text{ and } i \neq j \tag{17}$$

If autocorrelation points are removed (the correlation of data from one telescope station with itself), the number of visibility points, N_{uv} is given by Equation 18.

$$N_{uv} = N_{stations}\ \left(N_{stations} - 1\right) \tag{18}$$

Each uv point samples the Fourier transform of the sky brightness.[10] The more uniform the uv coverage is, the lower the undesirable side lobes in the synthesized beam of the array.[11, 12, 13] Cornwell[11] proposed a metric for the uv coverage as the distance between all uv points as shown in Equation 19. This metric is implemented as the an objective function in the

$$\text{Maximize } uv \text{ coverage} = \sum_{i,j,k,l} \sqrt{\left(u_{ij} - u_{kl}\right)^2 + \left(v_{ij} - v_{kl}\right)^2} \tag{19}$$

Another objective function that is usually considered is the minimization of the distance between the different stations as these distances represent the cost of the fiber optic cable needed to connect the stations. These are two opposing objectives because maximizing the uv coverage requires the stations to be placed as far as possible from each other. In this case study, only the uv coverage maximization metric is considered. Regardless of the number of stations, the solution of this single objective, unconstrained problem is known to place all available stations on a perimeter of a circle. The design variables then become the angles that describe the location the station on the circle. Consequently, the number of design variables is equal to the number of stations in the telescope array to be designed. In this research, four cases are investigated; these are: five, six, seven and eight station telescope arrays. The solutions for these cases are given in Reference 11, and were verified using an SQP approach (MATLAB's fmincon) for use in the PSO and GA effectiveness tests. These known solutions are shown in Figure 3 below.

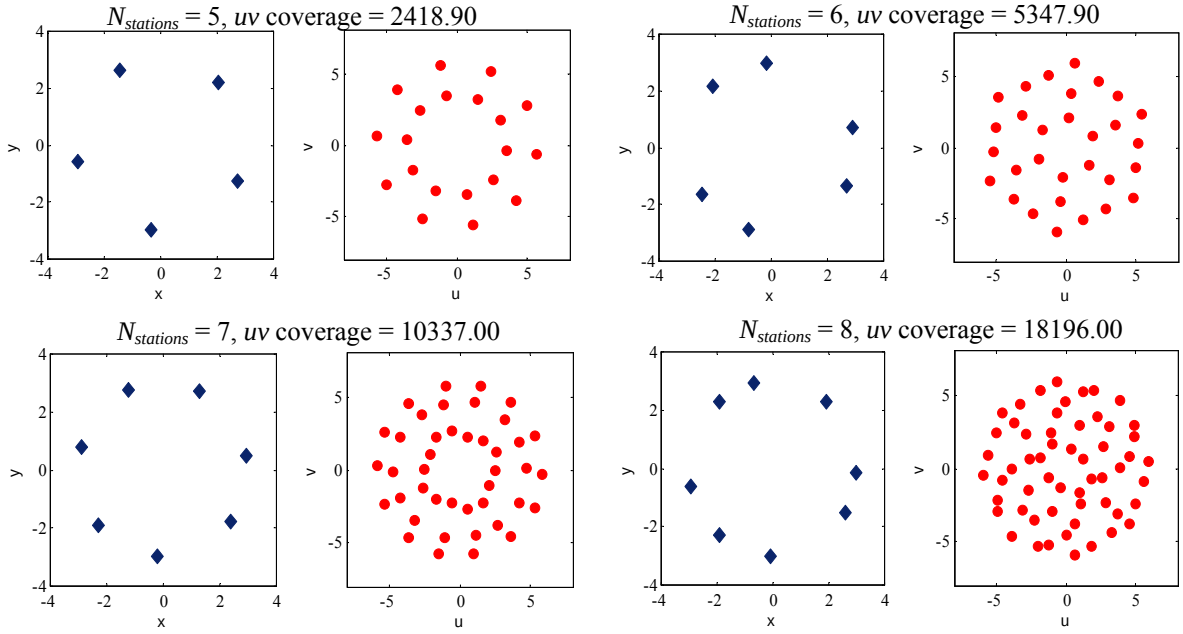

Figure 3. Optimal placement of telescope array stations and the corresponding *uv* coverage.

Communication Satellite Reliability-Based Design

This problem involves designing the payload and bus subsystems of a commercial communication Geosynchronous satellite with given payload requirements. The objective is to minimize the spacecraft overall launch mass, which is a surrogate for cost, given constraints on payload and system reliability. The problem also involves geometrical constraints imposed by the choice of the launch vehicle. The problem includes six functional constraints and 27 discrete design variables representing the technology choices and redundancy levels of the satellite payload and bus subsystems. For a complete description of the problem, see References 14, 15, and 16. The best solution found for this discrete problem is satellite with an optimal mass of 3415.8 kg. The above references describe the technology choices and redundancy levels of the different subsystems in this optimal spacecraft design.

Results and Discussions

Two performance tests were carried out for both algorithms under consideration, namely, PSO and the GA. The performance tests were carried out for eight sample problems: the Banana function, the Eggcrate function, Golinski's Speed Reducer, Reliability-based Satellite Design, 5-station, 6-station, 7-station and 8-station Radio Telescope Array Configuration. The first test is the effectiveness test, which measures the quality of the solutions found by the heuristic algorithm with respect to known solutions for the test problems. This test investigates whether the quality of the solutions obtained is greater than 99%. The second test is the efficiency test, which investigates whether the computational effort of PSO is less than that of the GA for the sample problem set using the same convergence criteria. These statements of the two tests were set as the alternative hypotheses while their complimentary statements were set as the null hypotheses, which is a traditional hypothesis testing approach. Both tests were conducted using acceptable Type I and Type II errors of 1% each. The *t*-statistic values that were calculated for the PSO runs and GA runs for all eight sample problems are summarized in Table 2 below. The sample problems below are organized in order of increasing number of design variables, a measure of complexity.

Table 2: Calculated *t*-values for the effectiveness and efficiency hypotheses tests

	Effectiveness Test, $t_{critical}$ = 2.0 Calculated *t*-value		Efficiency Test, $t_{critical}$ = 2.5 Calculated *t*-value
	PSO	GA	
1- Banana Function	8404.05	-2.49	1.9980
2- Eggcrate Function	312420.20	5214.34	7.2743
3- Telescope Array (5 Stations)	∞	∞	19.2995
4- Telescope Array (6 Stations)	∞	∞	16.8674
5- Telescope Array (7 Stations)	∞	∞	20.9451
6- Telescope Array (8 Stations)	∞	∞	23.7893
7- Golinski's Speed Reducer	-0.39	1251.28	10.3079
8- Satellite Design	-2.13	6.47	19.6993

The effectiveness tests for PSO and the GA show that $t > t_{critical}$ in most sample test problems. This leads to the rejection of the null hypothesis and the acceptance of the alternative hypothesis that the quality of the solutions of both approaches is equal to or greater than 99% in most of the test problems. The alternative hypothesis is accepted with a confidence level of 99%. The calculated infinity *t*-values in the telescope array problems are obtained because in each of the runs, PSO and the GA consistently found the known solutions for the 5, 6, 7, and 8 station array problems. Therefore, the quality of the ten solutions in each case is 100% and the standard deviation is zero. Figure 4 below summarizes the values implemented in the effectiveness test for both PSO and the GA.

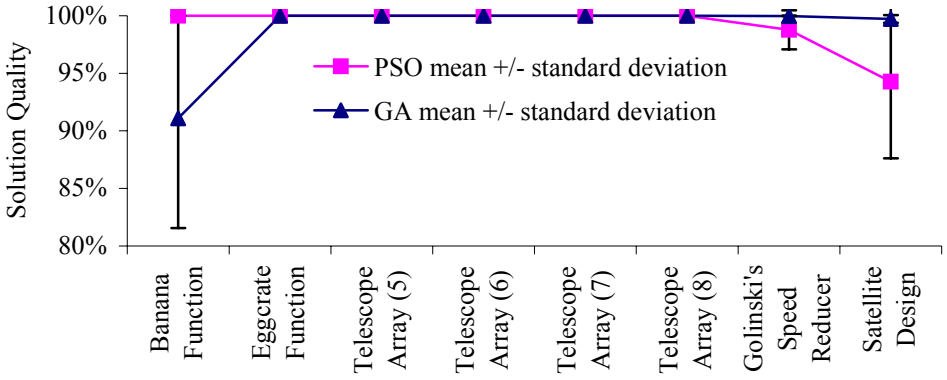

Figure 4. The mean quality of the solutions obtained by PSO and the GA for eight test problems.

In only one case, the Banana function, the null hypothesis is accepted for the GA runs, therefore it cannot be proved that the GA solutions have high quality for this problem. By further investigating the data, it is found that the mean quality of the GA solutions for this problem is 91% with a standard deviation of 9.5%. This mean is significantly lower than the 99% quality level required in the test. It is surprising that the GA could find high quality solutions for the rest of the test problems that involve continuous design variables and with larger number of design variables (problems 2 to 7) but was not able to obtain the same quality for the Banana function problem. A larger sample size might lead to improving the mean and standard deviation of GA solution quality for this problem.

PSO solutions for all test problems exceeded the 99% quality limits except for the last two problems. Further investigation of the data reveals that the mean quality of PSO solutions for Golinski's speed reducer is 98.8% with a standard deviation of 1.7%. The scatter in the data of the sample does not allow for achieving the quality limit of 99% to be met with sufficient confidence. As for the satellite design problem, it is expected that PSO does not perform well on this problem given the fact that all the design variables are discrete and that PSO is inherently continuous. The data in Figure 4 seems to suggest that there might be a new hypothesis to be tested. Namely that PSO performs well on small and medium sized problems, but scales poorly to larger problems. This new hypothesis will be tested in future work.

The efficiency test shows $t > t_{critical}$ for all test problems except for the Banana function. These results lead to the rejection of the null hypothesis and the acceptance of the alternative hypothesis with a confidence level of 99%. The interpretation of these results is that for all sample problems, except for the Banana function, the computational effort required by PSO to converge to a solution is less than that of the GA 99% of the time using the same convergence criteria. From Figure 5, it appears that the scatter of the GA runs for the Banana function is the reason behind the acceptance of the null hypothesis that PSO does not outperform the GA for the Banana function as it did in the rest of the test problems.

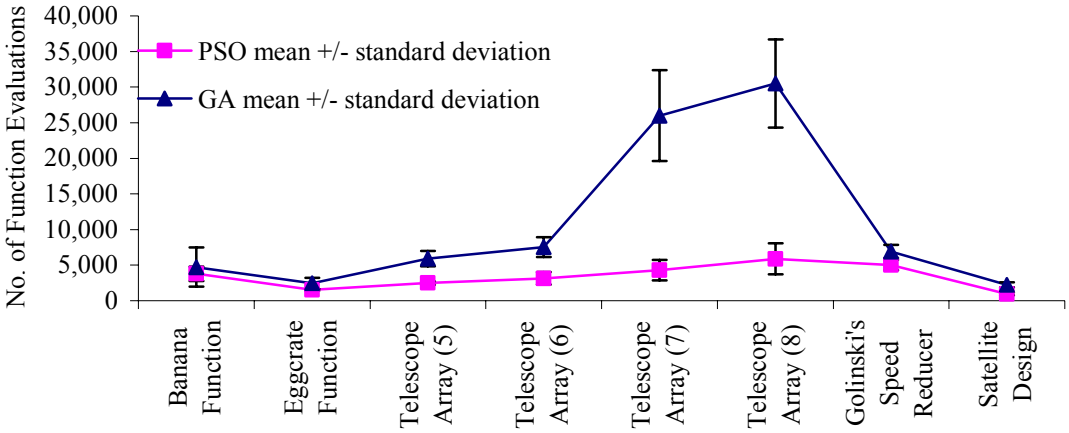

Figure 5. The mean computational effort of the solutions obtained by PSO and the GA using 10 run samples for eight test problems.

Figure 6 below couples the results of the effectiveness and efficiency tests. It shows the ratios of the mean values of the solution qualities and computational effort of the GA and PSO. The insight learned from Figure 6 is that for a range of test problems with variable complexity, PSO and the GA were able to arrive at solutions with the same quality, which is indicated by the flat line of a ratio of 1 for the means of the solution qualities. However, the varying ratios of the computational effort provides two conclusions: first, PSO does offer a less expensive approach than the GA in general, second, the expected computational savings offered by PSO over the GA is problem dependent. It appears that PSO offers more computational savings for unconstrained nonlinear problems with continuous design variables whereas the computational savings is lower for constrained and mixed integer nonlinear problems.

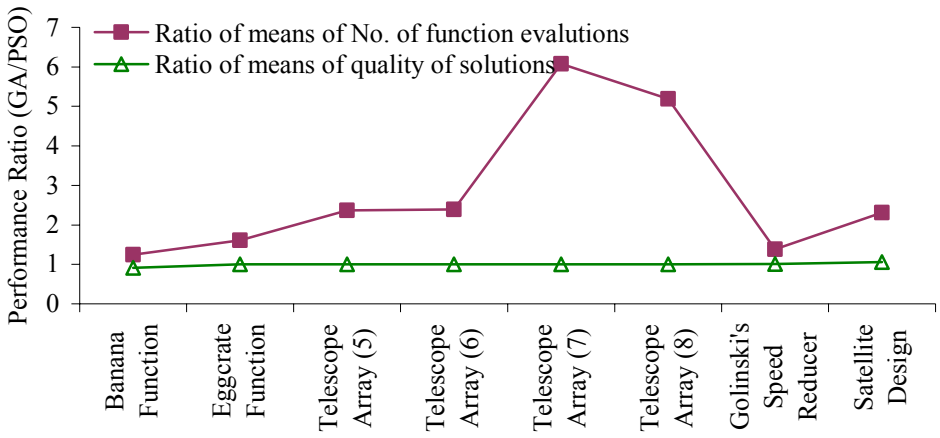

Figure 6. Comparison of ratios of solution quality and computational effort of PSO and the GA.

Conclusions

Particle Swarm Optimization (PSO) is a relatively recent heuristic search method that is based on the idea of collaborative behavior and swarming in biological populations. PSO is similar to the Genetic Algorithm (GA) in the sense that they are both population-based search approaches and that they both depend on information sharing among their population members to enhance their search processes using a combination of deterministic and probabilistic rules. Conversely, the GA is a well established algorithm with many versions and many applications. The objective of this research is to test the hypothesis that states that although PSO and the GA on average yield the same effectiveness (solution quality), PSO is more computationally efficient (uses less number of function evaluations) than the GA. To investigate this claim, two statistical tests were set to examine the two elements of this claim, equal effectiveness but superior efficiency for PSO over the GA.

To carry out the *t*-tests, eight sample test problems were solved using both PSO and the GA over multiple runs. The test problems include three well-known benchmark test problems; these are: the Banana (Rosenbrock) function, the Eggcrate Function, and Golinski's Speed Reducer. Two space systems design problems were also investigated to test the algorithms on real-life engineering problems. The first problem is the configuration of a ground-based multi-station radio telescope array. Four versions of this problem were investigated including 5, 6, 7 and 8-station arrays. The second test problem involves the reliability-based design of a commercial communication satellite. All test problems involve continuous design variables only except for the satellite design problem which contains discrete design variables representing subsystems technology choices and redundancy levels. The eight test problems represent a wide range of complexity, nonlinearity, and constraint levels.

Two metrics were identified for the two *t*-tests. The effectiveness test for both PSO and the GA uses a solution quality metric that measures the normalized difference between the solutions obtained by the heuristic approaches and known solutions for the test problems. The efficiency test uses the number of function evaluations needed by the heuristic approaches to reach convergence. The same convergence criterion is enforced on PSO and the GA.

The results of the *t*-tests support the hypothesis that while both PSO and the GA obtain high quality solutions, with quality indices of 99% or more with a 99% confidence level for most test problems, the computational effort required by PSO to find high quality solutions is less than the effort required by the GA to find the same high quality solutions. The results further show that the computational efficiency superiority of PSO over the GA is statically proven with a 99% confidence level in 7 out of the 8 test problems investigated. Further analysis shows that the difference in computational effort between PSO and the GA is problem dependent. It appears that PSO outperforms the GA with a larger differential in computational efficiency when used to solve unconstrained nonlinear problems with continuous design variables and less efficiency differential when applied to constrained nonlinear problems with continuous or discrete design variables. Researchers interested in furthering PSO development are encouraged to investigate the results presented in this paper.

References

[1] Kennedy, J. and Eberhart, R., "Particle Swarm Optimization," *Proceedings of the IEEE International Conference on Neural Networks*, Perth, Australia 1995, pp. 1942-1945.

[2] Kennedy, J. and Eberhart, R., *Swarm Intelligence*, Academic Press, 1st ed., San Diego, CA, 2001.

[3] Goldberg, D., *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, Reading, MA, 1989, pp. 1-25.

[4] Venter, G. and Sobieski, J., "Particle Swarm Optimization," AIAA 2002-1235, *43rd AIAA/ASME/ASCE/ AHS/ASC Structures, Structural Dynamics, and Materials Conference*, Denver, CO., April 2002.

[5] Williams, E. A., and Crossley, W. A., "Empirically-Derived Population Size and Mutation Rate Guidelines for a Genetic Algorithm with Uniform Crossover," *Soft Computing in Engineering Design and Manufacturing*, P. K. Chawdhry, R. Roy and R. K. Pant (editors), Springer-Verlag, 1998, pp. 163-172.

[6] Harnett, D., *Statistical Methods*, 3rd ed., Addison-Wesley, 1982, pp.247-297.

[7] Volk, W., *Applied Statistics for Engineers*, 2nd ed., McGraw-Hill, 1969, pp. 109-148.

[8] Web site http://mdob.larc.nasa.gov/mdo.test/class2prob4.html cited August 16th, 2004.

[9] Ray, T., "Golinski's Speed Reducer Problem Revisited," *the AIAA Journal*, Vol. 41, No. 3, 2003, pp. 556 -558.

[10] Thompson, A., Moran, J., and Swenson, G., *Interferometery and Synthesis in Radio Astronomy*, Wiley Interscience, New York, 1986.

[11] Cornwell, T., "Novel Principle for Optimization of the Instantaneous Fourier Plane Coverage of Correlation Arrays," *IEEE Transactions on Antennas and Propagation*, AP, Vol. 36, 1988, pp. 1165.

[12] Keto, E., "Shapes of Cross-Correlation Interferometers," *APJ*, Vol. 475, 1997, pp. 843.

[13] Cohanim B., Hewitt J., and de Weck O., "The Design of Radio Telescope Array Configurations using Multiobjective Optimization: Imaging Performance versus Cable Length", *The Astrophysical Journal*, Supplement Series, 154, 705-719, October 2004.

[14] Hassan, R., *Genetic Algorithm Approaches for Conceptual Design of Spacecraft Systems Including Multiobjective Optimization and Design under Uncertainty*, doctoral thesis, Purdue University, May 2004.

[15] Hassan, R., and Crossley, W., "Multiobjective Optimization of Communication Satellites with a Two-Branch Tournament Genetic Algorithm," Journal *of Spacecraft & Rockets*, Vol. 40, No. 2, 2003, pp. 266-272.

[16] Hassan, R., and Crossley, W., "Variable Population-Based Sampling for Probabilistic Design Optimization with a Genetic Algorithm," AIAA-2004-0452, *42nd Aerospace Sciences Meeting*, Reno, NV, January 2004.