Aerospace Design Optimization Using A Compound Repulsive Particle Swarm

by

Jeffrey Michael Badyrka

A thesis submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Auburn, Alabama
May 9, 2011

Abstract

This study compares contemporary design optimization algorithms for use in missile design applications. The engineering design problem for this study is the development of a desired single stage solid propellant missile system. The methods are used independently to match a prescribed set of parameters defining the missile's geometry and propellant characteristics. For this development, three different design optimization techniques are considered: a real-coded genetic algorithm (GA), a binary GA, and a Repulsive Particle Swarm Optimizer (RPSO). Since there is not nearly as much known about how the RPSO operates compared to the GA's, an extensive effort was invested in the study of how changing RPSO controlling parameters would affect its operation. Also, a new hybrid optimizer has been developed for this study involving a separate Particle Swarm imbedded within the standard RPSO. The algorithms are compared based on their speed and effectiveness in solving the design problem, with the figure of merit being a factor based on the desired performance goals for the missile.

Table of Contents

List of Tables

List of Figures

Nomenclature

μ	Mutation Rate
σ	Mutation Amount
ρ	Atmospheric Density
ρ_p	Propellant Density
A*	Nozzle Throat Area
A_b	Burning Area
Ae	Nozzle Exit Area
Apdly	Autopilot Delay Time
b2t	Tail Semi-Span
b2var	Nozzle Vane Semi-Span
b2w	Wing Semi-Span
C_D	Drag Coefficient
crw	Wing Root Chord
crt	Tail Root Chord
D	Drag
dele0	Initial Elevator Angle
diath	Diameter of Throat
dbody	Diameter of Missile Body
delx-y	Delta X for Y Corrections
delx-z	Delta X for Z Corrections
dtchek	Time Step to Actuate Controls

eps	Epsilon-Grain
f	Propellant Grain Fillet Radius
fn	Fractional Nozzle Length Ratio
GA	Genetic Algorithm
gainp1	Pitch Multiplier Gain
gainp2	Gain in Pitch Angle Difference
gainy1	Yaw Multiplier Gain
gainy2	Gain in Yaw Angle Difference
gl	Propellant Grain Length
kfuel	Propellant Fuel Type
lnos	Length of Missile Nose
nsp	Number of Star Points-Grain
ptang	Star Point Angle
r	Propellant Burn Rate
ratio	Nozzle Expansion Ratio
rbody	Radius of Body
rbi	Outer Radius of Grain
ri	Propellant Inner Grain Radius
rnos	Radius of Missile Nose
rp	Propellant Outer Grain Radius
RPSO	Repulsive Particle Swarm Optimizer
$\bar{S}$	Reference Area
t	Time

T	Thrust
thet0	Initial Launch Angle
tleswp	Tail Leading Edge Sweep Angle
TOF	Time of Flight
trt	Tail Taper Ratio
trw	Wing Taper Ratio
V	Velocity
V_f	Final Velocity
V_o	Initial Velocity
W	Weight
$\bar{W}$	Average Weight
W_f	Final Weight
W_o	Initial Weight
wleswe	Wing Leading Edge Sweep Angle
xcet	Nozzle Exit Diameter Ratio
xLEw	X-Location of Wing Leading Edge
xTEt	X-Location of Tail Trailing Edge

1 INTRODUCTION

Engineering design problems can generally be reduced to a set of crucial design variables that ultimately determine the effectiveness of a design. It is not a trivial matter to determine what values of these design parameters are required to match performance predicted by the objective function defining the effectiveness of a complex system. Manually varying all of these parameters can be an intractable task. Alternatively, optimization algorithms can be very effective in addressing a large class of these complex design problems. Using various mathematical schemas, optimization algorithms search the design space far more effectively and efficiently for solutions which meet the required design goals. The development of optimizers has become an ongoing area of research as their need and use has become more widespread. As optimizers are developed, it is important that they be tested on various problems and compared to other optimizers that have already been tested extensively and proven their merit. That is the motivation for this work: improving, developing, and testing optimizers for use in engineering applications.

In order to effectively evaluate optimization algorithms, a somewhat difficult, yet ultimately achievable goal must be set. The goal must not be too simple so as to allow any basic algorithm to reach a solution in a relatively short period of time, and yet not so complex as to require extended computational times or clusters of processors in order to evaluate the objective function. The design of a single stage solid propellant missile by manipulating key parameters required for full system performance modeling satisfies these requirements and is of current interest[1].

The modern collection of optimizers has grown to include algorithms based on evolutionary and social behavior models. Genetic Algorithms (GA's) have been used successfully in aerospace engineering applications for the optimization of spacecraft controls[2,3], turbines[4], helicopter controls[5], flight trajectories[6], wings and airfoils[7,8], missiles [1,9,10,11], rockets[12], propellers[13] and inlets[14] . Particle Swarm Optimizers (PSO's) have also been used for a variety of applications and research areas, including some engineering applications[15]. In some cases, real-coded GA's have been shown to outperform their binary coded counterparts[16,17,18]. These comparisons and subsequent results are the key motivators for the current effort. Increasing computing power has led to the development of increasingly more complex optimizers that are able to proficiently solve increasingly more complex design problems.

Genetic Algorithm's were originally developed from theories outlined by Professor John Holland. In his book "Adaptation in Natural and Artificial Systems"[19], Holland prescribed schemes for using population based adaptive optimizers. The key principle driving Holland's methodology is survival of the fittest. This is achieved by first creating a population of members representing candidate solutions. The performances of these members are then analyzed and compared to one another. The weaker (worse) solutions are killed off and the stronger (better) ones are left to "reproduce" and "mutate" to produce superior offspring. Subsequent populations are created using a variety of strategies, including the tournament style of evolution.

The Particle Swarm Optimization method is based on the application of the philosophy of bounded rationality and decentralized decision-making[20]. PSO works on the principles of social behavior observed in natural groups such as a swarm of birds or a school of fish[21]. The understanding of how members in these groups move and communicate with each other is essential to the operation of this optimizer. The members, or particles, in these swarms communicate with each other in order to see which ones have better solutions. Those with worse solutions will attempt to “move” towards those better solutions. Throughout this process, each particle will continue to search within its own area for a better solution.

The search for more efficient optimizers for use in aerospace and many other applications continues. Prolonged run times and the inability of some optimizers to sufficiently converge have led to the use of a diversified group of optimization algorithms. The application of multiple optimizers may also result in the development of completely different but nearly equally optimal solutions.

This study compares results for various optimization algorithms used for designing solid motor rockets. The study involves the use of Genetic Algorithms, a Repulsive Particle Swarm Optimizer, and a newly developed staged Repulsive Particle Swarm Optimizer. The algorithms are compared on their overall optimization result and on the speed of convergence to this result.

2 GENETIC ALGORITHM METHODOLOGY

There are two Genetic Algorithms used in this study that the Particle Swarm will be compared against. These two GA’s, a real-coded GA and the binary GA IMPROVE©[22], were used in a previous study[23] that will provide the foundation for the optimizer development done in this body of work. The GA’s will allow for a direct comparison to be drawn between their solutions and the Particle Swarm results.

Both the real-coded GA and the binary GA IMPROVE© function on the same basic premise. The evolution of the populations is driven by tournaments between the existing members. Each discrete population is composed of information sets called individuals, with each individual possessing a fitness and a chromosome. These individuals are representatives of potential solutions to the current problem. The chromosomes are composed of genes which are the GA variables specified in an input file. These GA’s use the tournament style selection process by selecting certain members, or parents, according to their fitness and using them to create offspring for the next population.

The real-coded GA compares randomly selected pairs of parents in the current population with the member having the better fitness surviving for use in reproduction. Two of the surviving parents are then mated using the crossover and mutation routines that mix and transform their genes to create new offspring[23].

The crossover and mutation processes ultimately determine how the parameter evolution process occurs. The crossover routines determine how individual genes are selected from the parents and mixed to create the offspring. Multiple crossover routines may be performed including Uniform, Single-point, and Blend X. Uniform crossover works by randomly selecting a gene, or design parameter value, from either parent 1 or parent 2. Therefore, the offspring may be composed of any combination of the parents' genes. Single-point crossover uses a random number (Rnd#) between 0 and 1 to define what percentage of the first parent's genes will be given to the offspring. For example, if there are 10 design parameters and the random number is chosen as 0.4, then genes 1-4 of the first parent and genes 6-10 of the second parent will belong to the offspring. The genes selected from the first parent always starts at parameter #1 and continues until the random percentage of parameters is satisfied, at which point, the rest of the parameters are chosen from the second parent. Blend X[24] operates by multiplying the absolute value of the difference between the parameters of parent 1 and parent 2 by a random number (Rnd#) between 0 and 1 and adding it to the smaller of the two values. That is, if the parameter values of parents 1 and 2 are 10.0 and 5.0, respectively, then the offspring's parameter value will be (10.0 – 5.0)*Rnd# + 5.0. Unlike the Uniform and Single-point routines, Blend X produces offspring completely unique from their parents.

Because none of these crossover routines can create an offspring with parameter values that are outside the range of its parents', genetic mutation is necessary for solution diversity. Thus, a Gaussian mutation routine that is controlled by two factors, mutation rate and mutation amount, is used to transform the offspring members after they are created. The number of genes mutated by the routine is determined by the mutation rate, μ. This term can be set between 0.0 and 1.0, with 1.0 resulting in a mutation of every gene. The mutation amount, σ, controls how much each gene is mutated and usually ranges from 0.005 (very little mutation) to 0.5 (high mutation). To fully mutate the offspring's genes, a Gaussian distributed random number with a zero mean and unit variance is used in conjunction with the value of σ as shown below.

For i = 1 to number of genes:

$$Offspring(i) = \sigma * [xmax(i) - xmin(i)] * gaussian\_random\_number + Offspring(i) \quad (1)$$

xmax(i) and xmin(i) represent the maximum and minimum values for the gene variables specified by the user on input. If the mutation of any of the genes results in a value that falls below the specified minimum or above the specified maximum, then that value is automatically set to that minimum or maximum.

The binary GA uses a generational approach, while the real-coded GA uses a steady state operational mode. The differentiating characteristic for these two modes is the number of members from a generation which must be evaluated for performance. The binary GA evaluates all of the members from the current population and determines their fitness. Each member of the population is then replaced by the tournament system through the crossover and mutation routines. The objective function is then used to evaluate all of the new members and the entire process is repeated until the maximum number of generations is reached.

By operating in steady state, the real-coded GA is only required to replace one member of the population through the tournament system and crossover and mutation routines once the initial population is evaluated. The objective function is used to evaluate only the new member, which is then used to replace the worst performer of the previous generation. This process is repeated until the maximum number of iterations is reached. The new member immediately enters the genetic pool, resulting in an immediate move toward the optimal solution. This may result in very fast and accurate convergence for single goal problems[25] but may suffer from a lack of diversity since the steady state operation does not allow for as many random guesses as the generational approach. The real-coded GA must rely more heavily on the mutation and crossover routines, which was why it was essential to include the Blend X option.

3 REPULSIVE PARTICLE SWARM METHODOLOGY

The Repulsive Particle Swarm Optimizer (RPSO) developed by Mishra[20] has been modified for use in this undertaking. Members of the RPSO mimic the social behavior of a swarm of individuals. The individual particles are composed of the variables specified in the code input file. The individuals exist in a multidimensional space, each acting as a particle with a position and a velocity. The individual particles travel through the solution space all the while remembering their overall best position. The swarm members relay information about desirable positions to each other and modify their own positions and velocities accordingly. The swarm members can communicate by either doing "swarm best", in which the best position ever seen by any one member is known to all of the other members or by doing "local bests", in which the best position seen by a particle in a specific neighborhood is known to the members of that neighborhood. The following equations show how the particle positions and velocities are updated with each iteration.

For i = 1 to number of iterations:

$$v_{i+1} = \omega v_i + c_1 r_1 \left(\hat{x}_i - x_i\right) + c_2 r_2 \left(\hat{x}_{gi} - x_i\right) \tag{2}$$

$$x_{i+1} = x_i + v_{i+1} \tag{3}$$

The individual particles have position, *x*, and velocity, *v*. The inertial constant is represented by *ω*. Successful choices for *ω* tend to be slightly less than unity. The constants c_1 and c_2 control how much the particles will be aimed towards the good positions. Both usually have values around unity. r_1 and r_2 are random numbers ranging from 0.0 to 1.0. The overall best position that a particle has seen is represented by $\hat{x}$ and the global best seen by the swarm is represented by $\hat{x}_g$. If local bests are used, $\hat{x}_g$ is replaced by $\hat{x}_L$.

In this study, a variant of the general PSO known as the Repulsive Particle Swarm (RPSO) is used. This method tends to be more effective in finding the global optimum in a complex solution space. However, the RPSO may take longer to find solutions to certain problems. The previous equations for particle position and velocity are modified slightly.

For i = 1 to number of iterations:

$$v_{i+1} = \omega v_i + \alpha r_1(\hat{x}_i - x_i) + \omega\beta r_2(\hat{x}_{hi} - x_i) + \omega\gamma r_3 z \tag{4}$$

$$x_{i+1} = x_i + v_{i+1} \tag{5}$$

As before, r_1, r_2, and r_3 are random number between 0.0 and 1.0. In this case, the inertial weight, ω, will range from 0.01 to 0.7. The position of a randomly chosen other particle is represented by $\hat{x}_h$, and z is a randomly chosen velocity vector. α, β, and γ are all constants. In instances when the routine gets stuck in local optimum, a chaotic perturbation may be introduced to both the position and velocity of some particles.

The RPSO used in this effort has been modified from Mishra's RPSO to allow the particles to have a wider local search ability. Each particle travels around in its immediate surroundings searching for a better solution. A parameter called *nstep* controls the local area in which the particle is allowed to search. This parameter makes the particles act in a way that more resembles the real life behavior of swarms.

The particles are allowed to learn from the other members of the swarm. In the extreme case, every particle will learn from the overall best member. This, however, is not always practical or realistic. Hence, in the practical algorithm, a particle is only allowed to learn from a select few of its neighbors. The communication structure that the particles exist in, or the way in which the particles learn from their neighbors, is known as the topology. Topologies used here include random, ring and random, and ring. Figure 1 contains simple illustrations of how the particles might be connected through the various topologies in the solution space.

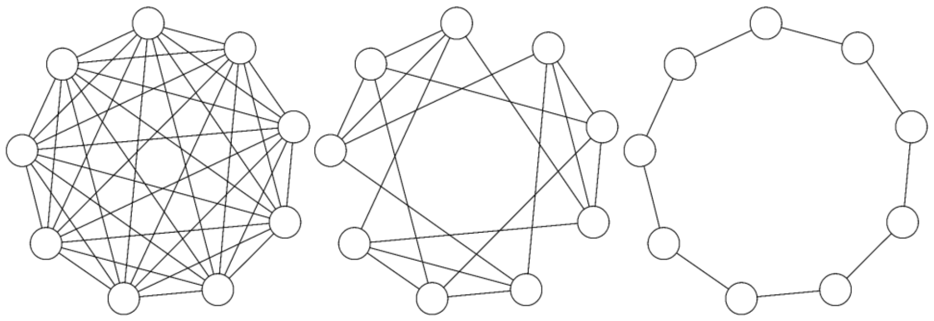


Figure 1. Particle Swarm Solution Space Topology[26]

Since a particle searches around within its own immediate area by itself, it may not learn much from interacting with only its closest neighbors. However, this RPSO may also be set so that a particle can learn from a predetermined number of randomly chosen members of the swarm. This method is seen to more closely reflect human social learning patterns. In this fashion, desirable characteristics filter through the population because there is a path to all of the population from any given member.

Further changes were made to the RPSO in order for it to work effectively with the solid motor missile systems code. Originally, the swarm was initialized with completely random positions and velocities. This caused problems with the missile code due to the fact that the problem is substantially constrained. Many of the initial members would be created with non-viable solutions (e.g. motor geometry conflicts, insufficient thrust for liftoff), which severely limited the swarms initial search ability and consequently, its overall search ability. To compensate for this, the RPSO was modified so that the initial population is created using only viable members. The optimizer runs through a prescribed number of iterations creating random parameter values for each member. Once the member has been evaluated, if it is determined to have a viable solution, it is kept and placed in the initial population. If not, the member is discarded and the iterations continue until the population is filled with the prescribed number of members or the number of iterations is completed. If the population is unable to be completely filled with viable solutions after the set number of iterations, then the rest of the members are created using random values, viable or not.

The implicit searching method of the particles and their somewhat chaotic movement within the solution space also caused problems for the constraints on the design space. The inherent nature of the RPSO causes many of the particles to "fly" outside of the prescribed parameter bounds. In some cases, particles will completely diverge from the swarm and the defined boundary of the solution space. Therefore, extensive effort was invested in "fencing in" the particles and preventing them from going outside of the parameter bounds. If the calculated particle velocity causes a parameter value to drop below the allowable minimum or go above allowable the maximum, a new velocity is calculated by multiplying the minimum or maximum allowable velocity, respectively, by a random number between 0.0 or 1.0. That is, the parameter value will be allowed to move in the same direction, but by only a fraction of the total distance to the boundary value.

One additional change was made to the existing RPSO to make it a more effective search tool. Previously, the particles would search in their local space at set intervals looking for better values. It was determined that it would be more reasonable, and eventually, more effective if the particles would search at random intervals while performing the *nstep* function in their local space. The source code for the modified RPSO can be found in Appendix D.

4 SOLID MOTOR SOUNDING ROCKET SYSTEM DESCRIPTIONS

A previous study was conducted in which the three optimizers were compared on their effectiveness to match a desired thrust versus time curve for a solid rocket motor[27]. The initial part of this study goes one step further and compares the optimizers based on their ability to design a desired sounding rocket with specific performance goals. The code creates preliminary design level simulations with complete geometry and burn characteristics for solid rocket motors. The code also calculates the weight of the structure used to encase the motor including the case, end cap, joints, and conical nozzle. The weight of the structure is then added to the propellant weight, igniter weight, and payload to determine the total weight of the system. The code uses a simple set of dynamic equations, including a set of empirical equations for drag, to fly the rocket on a vertical trajectory over the length of the motor burn[28]. When considering an object moving in the vertical plane, the sum of forces equations reduce to

$$T - D - W = \frac{W}{g}\left(\frac{dV}{dt}\right) \tag{6}$$

where T is thrust, D is drag, W is weight, g is gravity, and $\frac{dV}{dt}$ is the incremental change in velocity per change in unit time. By using small time steps, *dt,* the thrust can be considered to be approximately constant over the time step. The drag is defined as

$$D = \frac{1}{2}\rho V^2 C_D \bar{S} \tag{7}$$

with the drag coefficient, $C_{D,}$ calculated in terms of the current Mach number of the object. The weight is taken as the average weight over the time increment and is calculated as follows.

$$\bar{W} = \frac{W_o + W_f}{2} = W_o - \frac{r A_b \rho_p dt}{2} \tag{8}$$

A_b, ρ_p, and r are all characteristics of the solid propellant, while W_o is the initial weight of the rocket. By letting

$$C_1 = T - \bar{W} \tag{9}$$

and

$$C_2 = \frac{1}{2}\rho C_D \bar{S} \tag{10}$$

the sum of forces can now be written as

$$\frac{g}{\bar{W}} dt = \frac{dV}{C_1 - C_2 V^2} \tag{11}$$

When the above equation is integrated over the interval [V_o, V_f] for the case of $T > \overline{W}$, the equation becomes

$$\frac{2g\sqrt{C_1 C_2}\Delta t}{\bar{W}} = \ln\left[\left(\frac{C_1 + V_f\sqrt{C_1 C_2}}{C_1 - V_f\sqrt{C_1 C_2}}\right)\left(\frac{C_1 - V_o\sqrt{C_1 C_2}}{C_1 + V_o\sqrt{C_1 C_2}}\right)\right] = C_3 \tag{12}$$

and is further simplified by letting

$$C_4 = \frac{C_1 + V_o\sqrt{C_1 C_2}}{C_1 - V_o\sqrt{C_1 C_2}} e^{C_3} = \left(\frac{C_1 + V_f\sqrt{C_1 C_2}}{C_1 - V_f\sqrt{C_1 C_2}}\right) \tag{13}$$

From these equations, the final velocity can be calculated as follows.

$$V_f = \frac{C_1(C_4 - 1)}{\sqrt{C_1 C_2}(C_4 + 1)} \tag{14}$$

Once the final velocity is calculated, the average velocity can be determined and then multiplied by the time increment to determine the change in altitude over the time step.

For the case of $T < \overline{W}$, C_1 is negative, and the force equation then becomes

$$\frac{g\Delta t}{\bar{W}} = \frac{1}{a}\tan^{-1}\frac{Va}{C_1} \tag{15}$$

where

$$a = \sqrt{-C_1 C_2} \tag{16}$$

When integrated over the interval [V_o, V_f], the equation becomes

$$\tan\left(\frac{g\Delta t a}{\bar{W}}\right) = \frac{C_1 a(V_f - V_o)}{C_1^2 + V_f V_o a^2} = C_3 \tag{17}$$

The final velocity is now defined as

$$V_f = \frac{C_3 C_1^2 + C_1 a V_o}{C_1 a - C_3 a^2 V_o} \tag{18}$$

and the change in altitude of the vehicle calculated as stated previously.

The code is composed of 11 critical design variables that describe the geometry of the motor and nozzle[29]. The various optimizers are used to evaluate these solid motor parameters in order to match desired performance goals for the missile. The optimizers can be used to match any combination of three separate goals: burnout altitude, burnout velocity, and takeoff weight.

The first task of the program is to read in an input file containing mathematical constants, grain parameters, and goal parameters. Once the initial files are read, the optimizer's input file containing the variables to be optimized is read. For each new member of a population in the case of the GA's, or for each particle move in the case of the RPSO, a new missile grain design is constructed using data from the initial constants input file and parameters generated by the optimizer. The program then designs and burns the motor using a series of subroutines that determine geometry and burn characteristics. The program then uses the propellant parameters to determine the thrust of the motor as a function of time at sea level conditions. The specified grain geometry governs the part of the thrust equation that is independent of atmospheric pressure. The thrust and the calculated weights are then used with the dynamic equations to determine the overall performance of the missile. Figure 2 contains the general flow chart for the sounding rocket code.

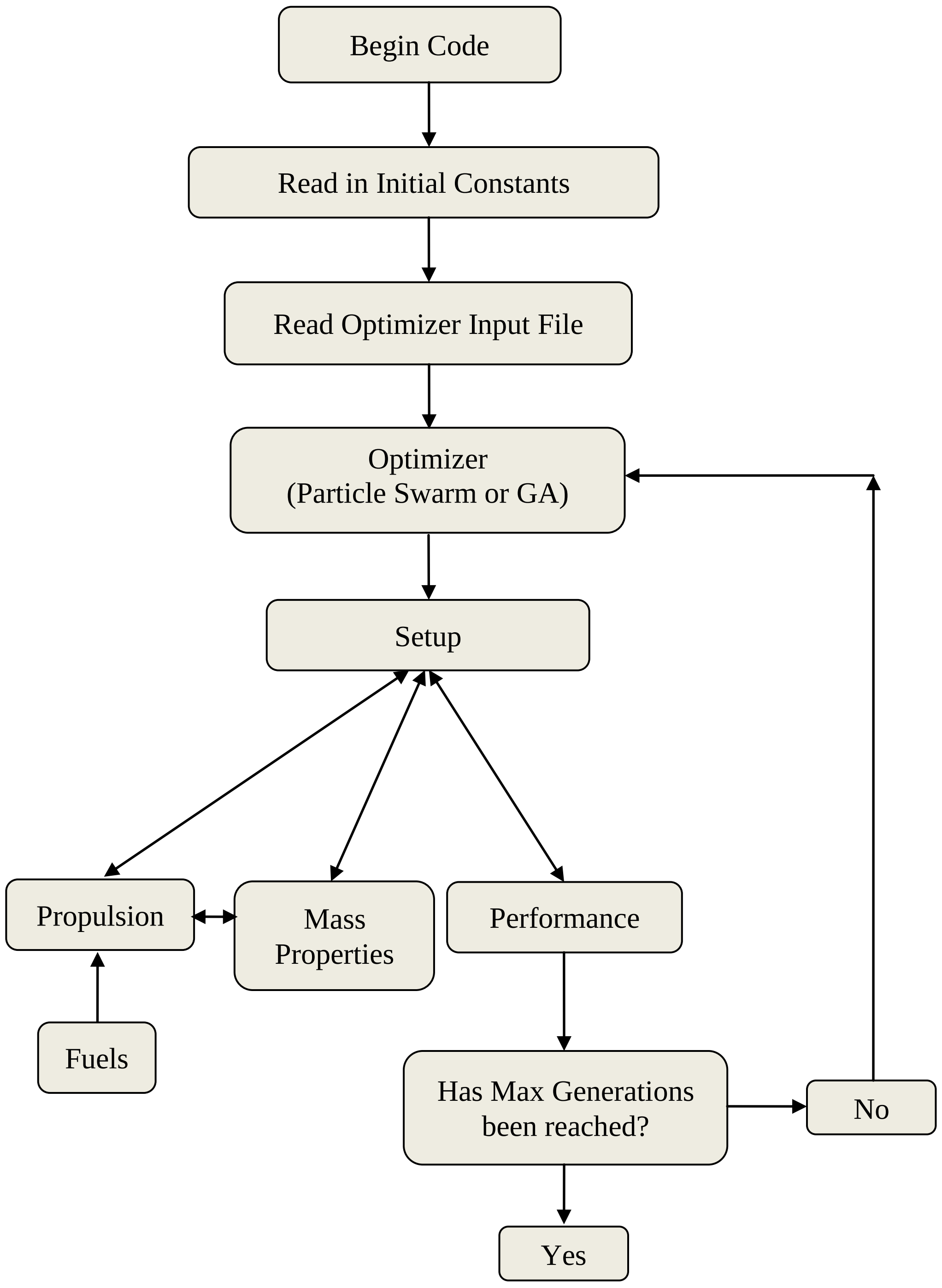


Figure 2. Sounding Rocket Design Code Program Flow

There are 11 design variables that the sounding rocket code uses to produce a physical model of the system. The variables are described in Table 1.

Table 1. Solid Motor Variable Definitions

Description	Variables
Propellant fuel type	kfuel
Propellant outer radius ratio	Rpvar=(Rp+f)/Rbody
Propellant inner radius ratio	Rivar=Ri/Rp
Number of star points	Nsp
Fillet radius ratio	fvar=f/Rp
Epsilon - star width	eps
Star point angle	ptang
Grain length	gl
Outer radius of grain	Rbi
Nozzle Throat Diameter	diath
Nozzle Expansion Ratio	ratio

Figure 3 has been adapted from Reference 30 to illustrate how these variables are used by the optimizer in conjunction with the initial constants to generate the solid propellant grain. For the solid motor system, circularly perforated grains, star grains and wagon wheel grains are possible with this geometric combination.

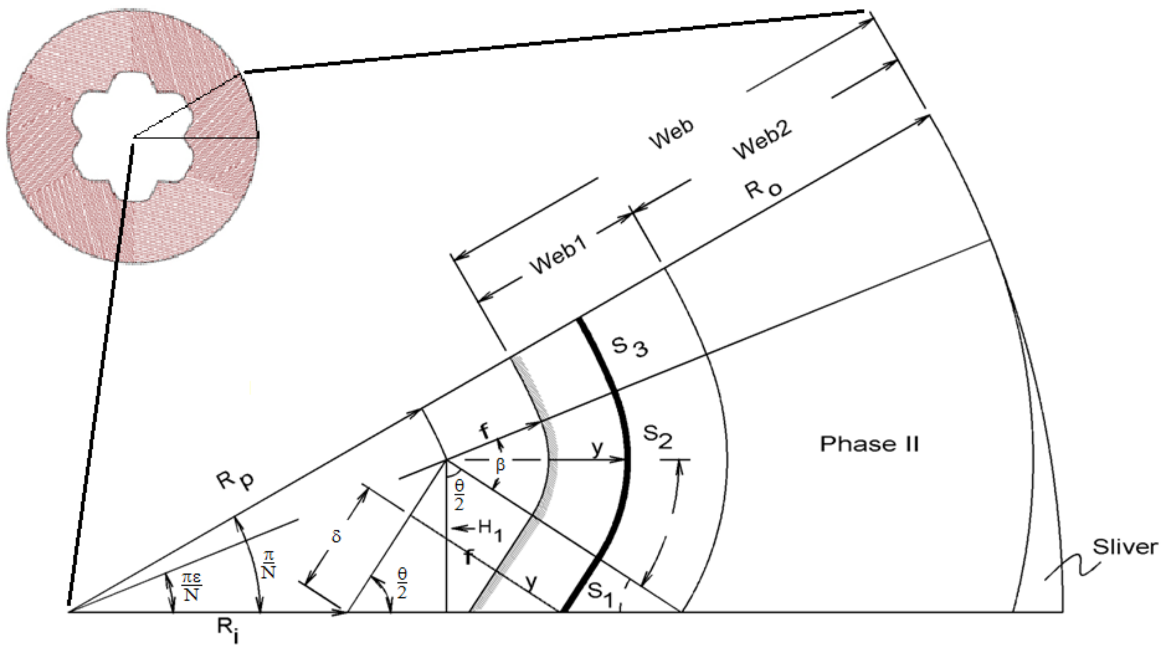


Figure 3. Solid Propellant Grain Cross-Section Schematic

5 PARTICLE SWARM INPUT PARAMETER TESTING

5.1 Particle Swarm Control Parameter Optimization

With the previous particle swarm optimization studies, there were no real guidelines regarding what the values of the input parameters controlling the optimizer should be. For the Repulsive Particle Swarm, this includes values for population size, neighborhood size, *nstep*, α, β, γ, and ω. For the optimization study focused around matching thrust versus time curves, arbitrary values were chosen for α, β, γ, and ω, while a limited amount of effort was put forth to find values for population size, neighborhood size, and *nstep* that seemed to produce the most efficient optimization results.

In order to fully evaluate the optimization potential of the Repulsive Particle Swarm Optimizer, it was necessary to study the RPSO's controlling parameters. It was not immediately clear what effect changing these parameters would have on the overall behavior of the optimizer, so a parametric study was performed using varying values of the particle velocity control parameters α, β, γ, and ω. A variable matrix was set up using these four parameters in which they were all allowed to vary between nine different values, thus creating a matrix of size 9^4 or 6561 total combinations. The allowable values of α, β, and γ were set to 0.05, 0.125, 0.25, 0.375, 0.50, 0.625, 0.75, 0.875, and 0.95, while the allowable values for ω were 0.02, 0.105, 0.19, 0.275, 0.36, 0.445, 0.53, 0.615, and 0.70. These quantities approximate an even distribution across the total range of allowable values for the parameters.

For the input parameter sets to be accurately compared, a penalty function had to be introduced into the optimizer routine. The intrinsic properties of the particle swarm cause many of the particles to tend to fly outside of the parameter bounds defined by the problem. When this happens, the particle is artificially reined in by the program and allowed to only move a randomly selected distance within the parameter bounds. Since certain combinations of the input parameters α, β, γ, and ω may tend to cause more particles to try to escape the bounds of the program, these sets may benefit purely on the basis of the random placement of the particles from the boundary control system. To prevent this, when a particle attempts to overstep the bounds of the program, it is still moved to a random place within the parameter bounds, but it is assessed a penalty in the form of a very high fitness value and not allowed to search locally after it has been moved.

For these RPSO tests, the solid motor sounding rocket code was used to carry a payload of 70lb to a burnout altitude of 50,000ft and a burnout velocity of 1,000ft/sec with a minimum total takeoff weight. Each test run was allowed to perform 250,000 function evaluations. The sounding rocket code was used for this study because it is complex enough to fully test the optimizer's ability, yet simple enough to provide solvable problems that did not require extended run times that would make it impractical for the number of runs needed to evaluate the input parameters. The fitness for each of the optimizer runs was calculated as follows:

$$Fitness = abs(Altitude - DesiredAltitude) + abs(Velocity - DesiredVelocity) + \frac{Weight}{1000} \quad (19)$$

The results of the RPSO input parameter grid search are shown in Figure 4.

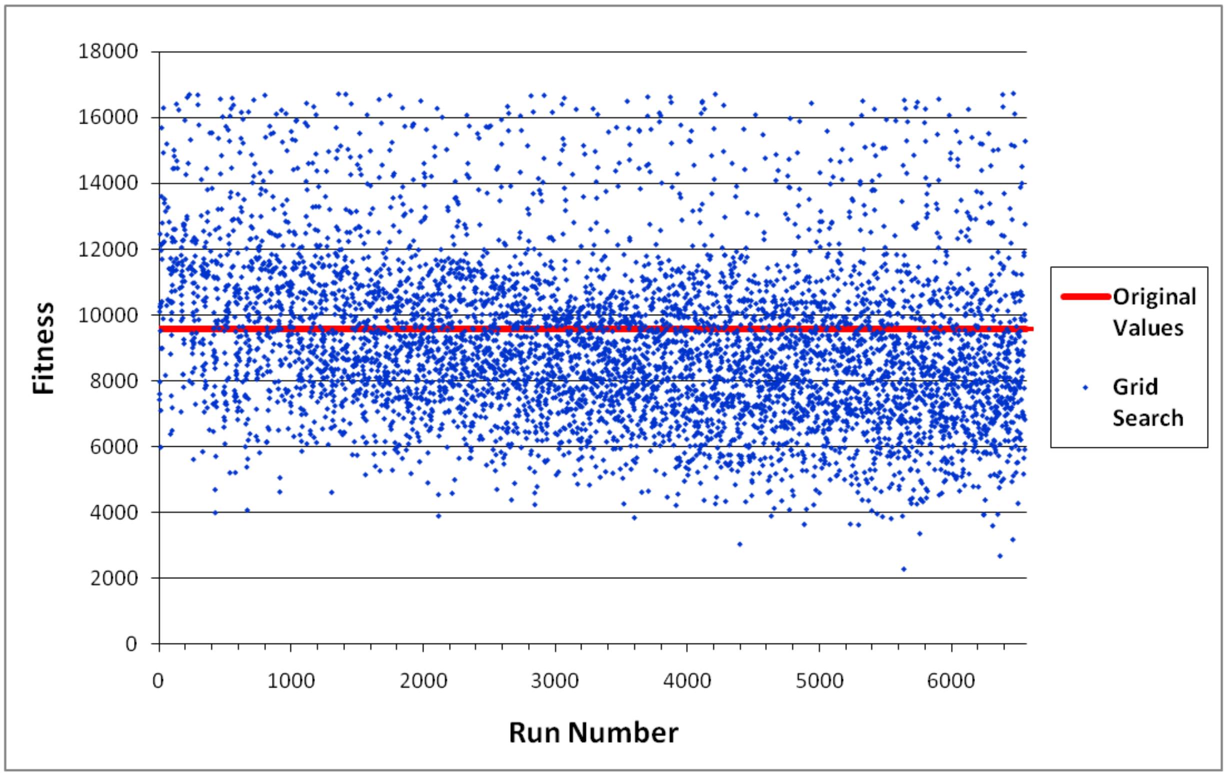


Figure 4. RPSO Sounding Rocket Parameter Testing Results

The matrix of variable parameter inputs for the RPSO produced a wide range of final fitness values for the optimization goal. Interestingly, the majority of the input sets produced fitness answers that fell below the red line, which represents the fitness achieved from using the original values of α, β, γ, and ω used in the prior study. It appears that there is a negative trend between the fitness values and the increasing run number. This might directly reflect the trend of increasing the first term of the parameter matrix, α. The best, or lowest, fitness value came from run number 5639. The values of the input variables and resulting fitness for the best performer of the grid search and the original set are shown in Table 2.

Table 2. Sounding Rocket RPSO Input Parameter Optimization Results

	Original Values	**New Best Values**
α	0.25	0.875
β	0.25	0.75
γ	0.50	0.625
ω	0.25	0.36
Fitness	**9578**	**2300**

All of the parameter values for the best performer of the grid search are different from those of the original settings. More specifically, the new values place a higher importance on α, or the best the particle has ever seen. The optimizer also performed better by increasing the effect of β, or the best value any neighbor particle has seen. The term γ, which adds more randomization into the particle movement, proved to work better with a higher value and ω, a repulsive term, seemed to work better with a slightly larger value. With the new values for the particle velocity parameters, the optimizer was able to decrease the previous fitness value by over 75%. This represents a significant improvement in fitness value for this goal set and hopefully, overall optimizer effectiveness.

Since the optimizer performed better with higher values for each of the four particle velocity parameters, it was deemed important to see what effect changing the values of the neighborhood sample size, *nn*, and the local search parameter, *nstep*, would have on the RPSO performance. With the total population containing 30 particles, *nn* and *nstep* were allowed to vary in increments of 5 from 5 to 30 and 5 to 15, respectively. Once again, all test runs were to perform 250,000 function evaluations. Table 3 contains the resulting input matrix for the values of *nn* and *nstep* and the corresponding fitnesses returned from the sounding rocket program.

Table 3. Secondary RPSO Input Parameter Optimization Results

nstep / *nn*	**5**	**10**	**15**
5	6707.40	7870.98	6351.36
10	2300.14	8852.32	6928.61
15	7402.22	6132.84	7586.63
20	9179.19	5913.00	7228.87
25	6704.24	5002.58	6505.95
30	8999.98	7734.87	6525.35

The best fitness resulted from using an *nstep* of 5 and an *nn* of 10, which happened to be the default values used during the previous studies. The next closest fitness value was over twice as large, however. Interestingly, having a value of 5 for *nstep* generally produced worse fitness values than for *nstep* equal to 10 or 15, except when paired with the value of 10 for *nn*. Conversely, there was no value of *nn* that generally produced either a better or worse fitness for all values of *nstep*. These characteristics might have resulted from having already optimized the values of α, β, γ, and ω with this pairing of *nstep* and *nn* as part of the input settings, or this pairing might actually be the best overall combination for the sounding rocket program. To fully address this question, further testing is required in which a complete input parameter matrix is examined with all possible combinations of α, β, γ, ω, *nstep*, and *nn*. Using the parameter values in this study, that would require 118,098 total optimizer runs, which would be extremely time consuming and not entirely practical. A more sensible approach would be to use another optimizer, either one of the GA's or a gradient optimizer such as the Pattern Search, to manipulate all of the RPSO control parameters in order to obtain the set that most efficiently uses the optimization algorithm.

5.2 Sounding Rocket Optimizer Comparison

Even though a significant improvement was made in the overall performance of the RPSO in the context of the sounding rocket problem, it does not necessarily mean that the RPSO is an effective optimizer. In order to fully evaluate its efficiency, it must be compared to the binary and the real-coded Genetic Algorithms, which are well established optimizers for this class of problems. Parameter tests were not performed on the two GA's in this study. Parameters that have been found to be successful in previous studies were used in these tests and held constant throughout. These parameters can be seen in Appendix A for the binary GA and Appendix B for the real-coded GA. Figure 5 shows a comparison of the convergence histories for the RPSO with its original input parameters, the RPSO with its newly optimized input parameters, and both GA's for the same three goal sounding rocket problem.

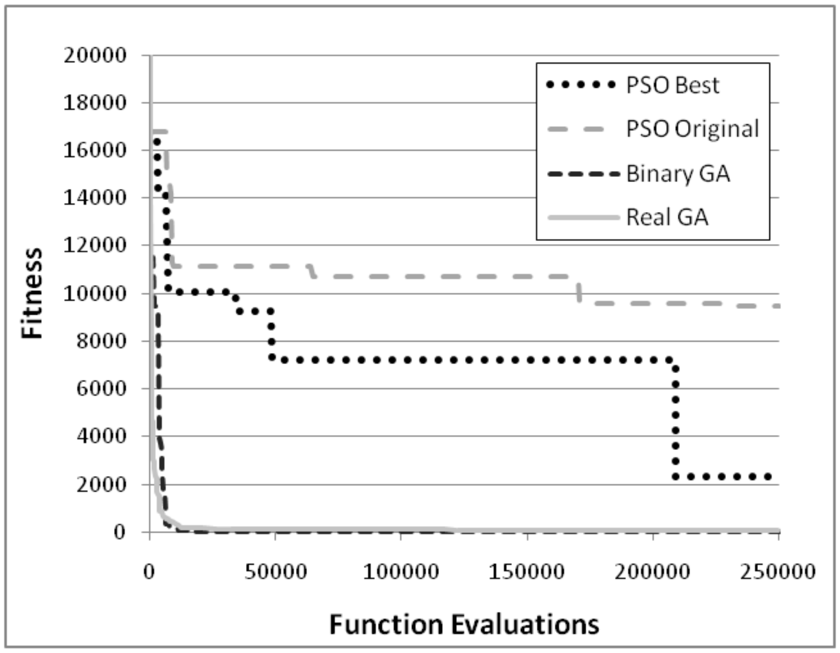


Figure 5. Sounding Rocket 3 Goal Convergence History

It is obvious that both of the GA's completely outperformed the RPSO with the binary GA being the top performer. Not only did the GA's develop solutions with much better fitness values, but they were also able to achieve this with a relatively small number of objective function calls. Although the improved RPSO surpasses the original RPSO very quickly, it does not make a significant improvement in fitness value until after around 200,000 function evaluations, at which point the fitness improves by over 75%. If allowed to run a bit longer, the RPSO might make another significant step towards an optimal solution that would rival those of the GA's, but this only shows how much less efficient it is for this particular problem.

The final motor design parameters and goal results are shown in Table 4 while the corresponding missile grain cross sections are shown in Figure 6. It is apparent that the two GA's and the two RPSO's all converged on very different motor designs, with the design from the original set of RPSO inputs being the most unlike the others, which would be expected. Looking at the motor cross section images, the RPSO appears to be approaching a design that is very similar to that of the binary GA, however, upon closer inspection, most of the parameter values are much different. If allowed to run for a greater number of function evaluations, the RPSO might produce a design completely different yet near equally optimal solution when compared to the GA's.

Table 4. Sounding Rocket Converged Design Parameters

PARAMETER	RPSO BEST	RPSO ORIGINAL	BINARY GA	REAL GA
propellant type	7	7	8	8
propellant RPVAR: (Rp+f)/(body radius)	0.52232459	0.53092006	0.53503937	0.75481233
propellant RIVAR: Ri/Rp	0.01010000	0.93402695	0.22137256	0.01000000
number of star points	8	13	11	17
fillet FVAR: f/Rp	0.06356675	0.18598332	0.04064516	0.01470821
epsilon star width	0.47259510	0.88108498	0.68666667	0.34154324
star point angle	1.27535397	6.47795456	5.67716550	8.90955565
grain length	239.78698141	271.08171369	301.36987000	241.53865028
outer radius of grain	23.76000000	23.76000000	22.95238100	19.06190103
throat diameter	19.80000000	19.80000000	12.86399300	19.49394728
nozzle expansion ratio	1.91562031	3.37402866	3.84251950	3.24619152
Altitude @BO	**47760.86**	**40958.55**	**50004.55**	**49999.98**
Velocity @BO	**960.67**	**600.84**	**997.71**	**968.22**
Initial Weight	**21672.41**	**22759.92**	**26505.14**	**10339.07**
Total Fitness	**2300.14**	**9463.37**	**33.34**	**42.14**

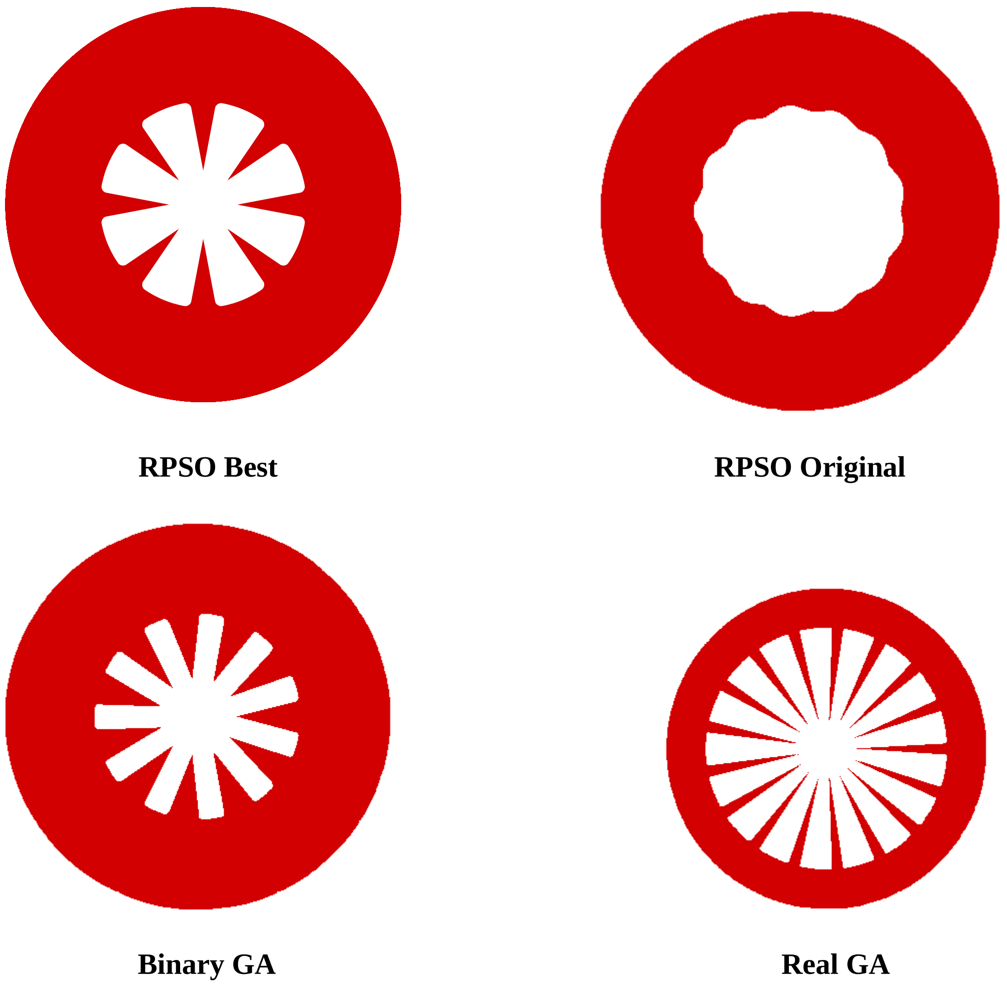


Figure 6. Sounding Rocket Final Grain Cross Sections

When comparing the designs based on the individual goals, the best performing RPSO made vast improvements in matching the desired burnout altitude and velocity while still making a modest improvement in decreasing the takeoff weight. The large fitness value for the RPSO is due mostly to it missing the desired burnout altitude by over 2,200ft, but this is only a difference of a little less than 5 percent.

The main goal of this study was to see if the performance of the Particle Swarm Optimizer could be improved by manipulating the control parameters of the algorithm.

Significant improvement was made in the optimizer's performance by just using a course grid to evaluate different combinations of the input parameters.

Even though the RPSO did not perform as well as the GA's for this problem, it still showed potential to serve as a viable optimization tool. It might be useful to use the RPSO in conjunction with the GA's to provide solution diversity when multiple nearly optimal solutions are needed. It is also important to note that the optimizers were only evaluated on one problem and goal set. The RPSO may perform just as well or even better than the GA's if it were only trying to match one goal, a different set of performance goals, or even used to solve different engineering problems. However, different types of problems may not yield as good of results with the current set of input parameters and may require their own optimized parameter sets. Other studies have shown that the RPSO, or a derivative thereof, has performed better than the GA on matching a curve to data[31].

6 COMPOUND REPULSIVE PARTICLE SWARM DEVELOPMENT

6.1 Hybrid Optimizer Methodology

It is apparent from the previous studies that the Particle Swarm generally converges much slower than the Genetic Algorithms for solid rocket motor preliminary design level trade studies. The development of this hybrid optimizer was pursued in an attempt to improve the RPSO’s ability to converge quickly to a good solution and possibly improve the overall fitness of the solution for this class of design problems. Hybrid optimizers incorporate the combination of two or more optimizers, usually of different types. A previous study combined the RPSO with the Pattern Search gradient optimizer and produced very promising results[31]. This study involves the integration of one RPSO into another, creating a compound RPSO.

The RPSO’s slow convergence may be due to the large solution space and the way the search mechanisms of the algorithm work. The particles cannot effectively search within a small area of the solution space even though they communicate with each other and perform individual local searches. The attachment of a second RPSO is used to more effectively search within an area around the current global best using a “mini” swarm.

The traditional RPSO can be thought of as a transport system for the mini-swarms. If the overall swarm best has improved after a generation, the mini-swarm search is initialized. This imbedded RPSO functions in the same way as the single-phase RPSO except that it is much more constrained. Similar to the parent swarm, the mini-swarm is initialized with a prescribed number of individuals with viable solutions. However, an overpopulation scheme has been implemented to help the performance of the mini-swarm. In this scheme, an oversized random population is created with the number of members being much larger than what will eventually be used in the mini-swarm. The members' fitnesses are compared and only a select number of members with the lowest fitnesses are chosen to continue in the mini-swarm. This scheme was implemented to help the members of the mini-swarm start closer to the optimal value, while not requiring a significant number of extra function evaluations.

The overall number of particles used in the mini-swarm is significantly less than that of the parent swarm, and they are only allowed to search within a small percentage of the solution space centered around the current best position. The global best used in the mini-swarm velocity equations is initialized with the current overall best and remains set as such until one of the mini-swarm members finds a better solution. The mini-swarm continues searching until the specified number of iterations is met, which is also significantly less than the large swarm. The input file for the mini-swarm can be seen in Appendix E.

A second modification has been made to the RPSO to help speed up convergence and improve the obtained solution. Since the RPSO tends to go long periods without finding better solutions, a counter has been set up so that if the optimizer has gone a prescribed number of iterations without improving, the mini-swarm optimizer is called once again to search around the current best solution. The hybrid optimizer methodology is represented in Figure 7.

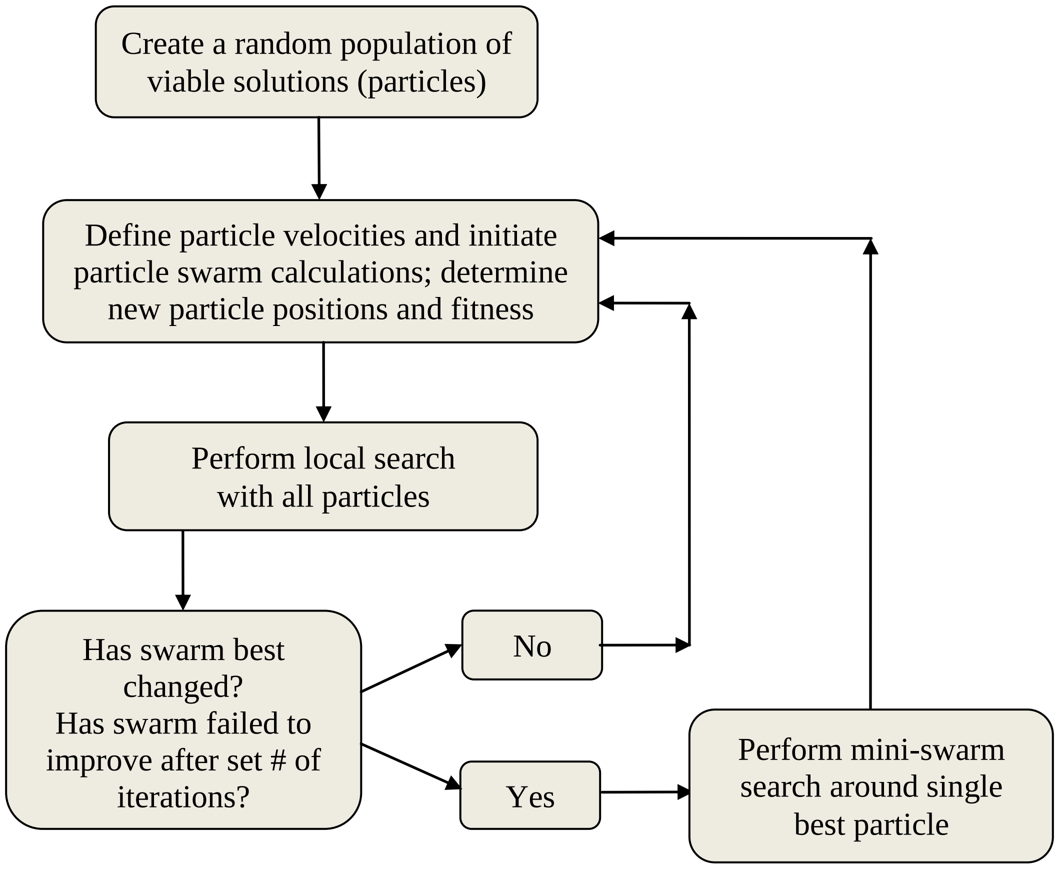

Create a random population of viable solutions (particles)
Define particle velocities and initiate particle swarm calculations; determine new particle positions and fitness
Perform local search with all particles
Has swarm best changed?
Has swarm failed to improve after set # of iterations?
No
Yes
Perform mini-swarm search around single best particle

Figure 7. Integrated Hybrid Optimizer Logic

6.2 Hybrid Optimizer Comparison

For initial testing, the compound optimizer was used on the same solid rocket motor code used in Reference 27. The results from these tests looked very promising and the hybrid optimizer was able to outperform the conventional RPSO in all test cases. These initial findings successfully demonstrated the compound optimizer's potential to work on more complex engineering problems.

For the next level of testing, the RPSO-RPSO compound optimizer has been attached to the same sounding rocket code used in the previous section to see if further improvement can be made in the performance of the particle swarm algorithm. The hybrid optimizer will be compared against the standard RPSO using the same goal set and number of function evaluations used for the sounding rocket in the previous study. Initially, the same parameter settings found for the best performer in the previous RPSO study were used for the parent swarm and the mini-swarm of the hybrid. Results from these settings for the sounding rocket code, however, did not prove as promising as the results from the solid rocket motor code used in Reference 27. At this point, it was decided to use another set of velocity parameters α, β, γ, and ω for the mini-swarm settings. The new set of parameters chosen was one used in the previous grid run search for the conventional RPSO. Although this set of parameters did not produce a final answer as good as the optimal set for the standard RPSO, it was primarily chosen for its fast convergence and relatively good fitness value.

This new combination of parent and mini-swarm parameters proved to work much better for the compound optimizer. Combinations of 5 and 10 particles allowed to search within 5% and 10% of the solution space were the settings used to test the mini-swarm. The convergence histories for these four hybrid settings and the previously optimized RPSO are shown in Figure

8. The numbers represented in the legend for the hybrid optimizer (ex. 5 – 5%) represent the number of particles within the mini-swarm and the percentage of the parameter range they are allowed to search within.

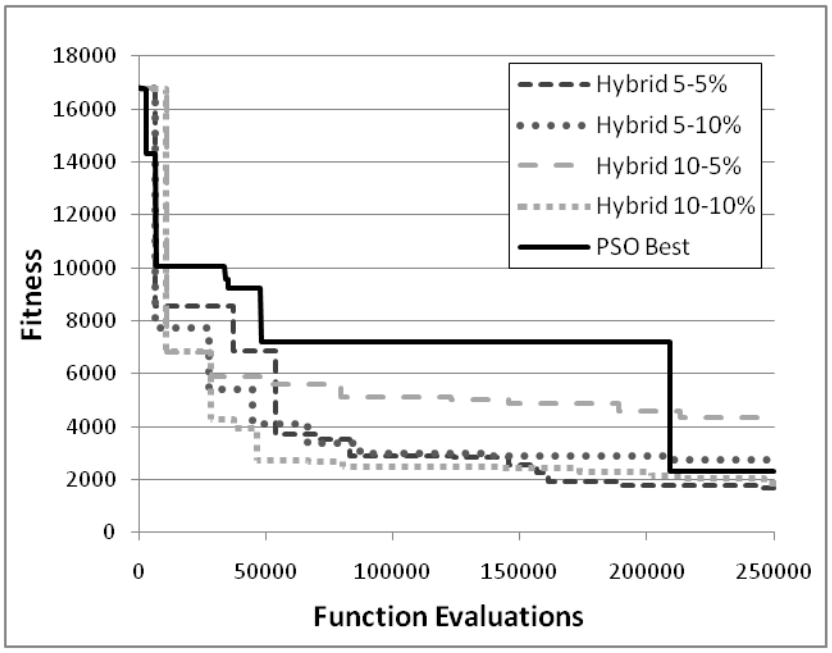


Figure 8. Hybrid Optimizer Sounding Rocket Convergence History

The compound optimizers behave much differently than the standard particle swarm. The hybrid particle swarm tends to converge much faster and in a much more continuous fashion. Until making the large fitness jump after the 200,000 function evaluation mark, the non-hybridized particle swarm was being beaten by all of the different hybrid input combinations.

Both the 5 particle-5% search area and the 10 particle-10% search area mini-swarm combinations performed very well for the hybrid. At the 150,000 function evaluation mark, they had already converged to answers which are comparatively close to the answer that the standard RPSO required an extra 50,000 function evaluations to achieve. Although the best performer of the hybrid runs performs more efficiently than the original, it still does not converge as well as the GA’s.

Table 5 shows the design parameter and goal values while Figure 9 shows the grain cross-section plots for the best performer of the hybrid and the single phase RPSO with the optimum input parameter set. The hybrid optimizer developed a somewhat different looking motor with a significantly better fitness than the standard particle swarm. The hybrid ends up matching the burnout altitude much better but does a worse job in matching the burnout velocity and minimizing the weight. This is most likely a direct result from how the fitness is actually determined and the weighting of the three goals. The altitude goal is of an order of magnitude higher than both of the other goals, thus it has the highest impact on the overall fitness.

The hybrid produced a design roughly 25% better than the standard particle swarm, and it did so with significantly fewer function evaluations. The mini-swarm allows the RPSO to search within more confined areas of the solution space which, consequently, allows the optimizer to converge faster and in a more gradual fashion. The required number of particles of the mini-swarm and the size of the solution space they are allowed to search in may be different for each problem, depending on the makeup of the solution space. This initial test of the compound RPSO hybrid proves promising, yet the optimizer requires further validation to ensure that it is a useful optimization device.

Table 5. Original and Compound RPSO Sounding Rocket Converged Design Parameters

PARAMETER	RPSO BEST	Compound RPSO (5 – 5%)
propellant type	7	7
propellant RPVAR: (Rp+f)/(body radius)	0.52232459	0.48779950
propellant RIVAR: Ri/Rp	0.01010000	0.43874862
number of star points	8	9
fillet FVAR: f/Rp	0.06356675	0.05419492
epsilon star width	0.47259510	0.52420674
star point angle	1.27535397	6.41346202
grain length	239.78698141	241.09208069
outer radius of grain	23.76000000	23.97510469
throat diameter	19.80000000	19.35065689
nozzle expansion ratio	1.91562031	2.78466686
Altitude @BO	**47760.86**	**48364.79**
Velocity @BO	**960.67**	**1051.18**
Initial Weight	**21672.41**	**22611.68**
Total Fitness	**2300.14**	**1709.00**

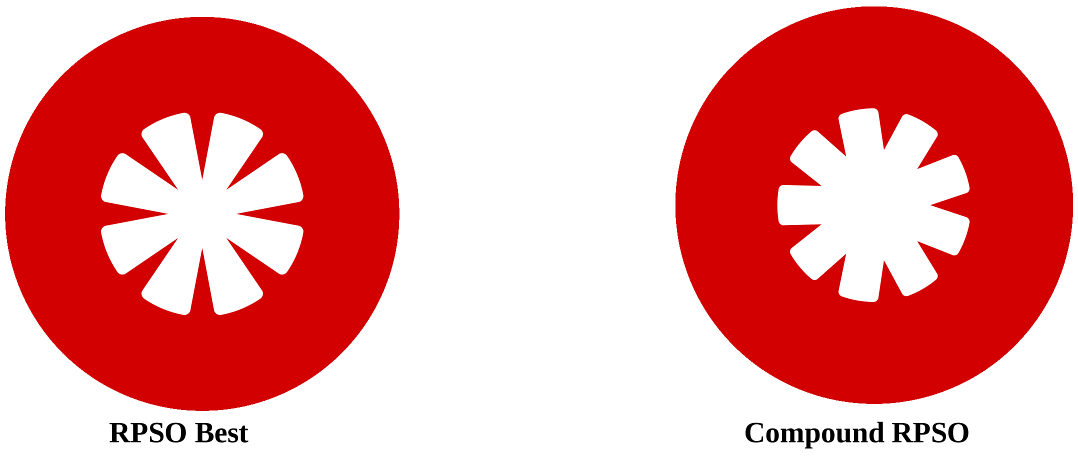


Figure 9. Original and Compound RPSO Sounding Rocket Final Grain Cross Sections

7 SINGLE STAGE SOLID MISSILE SYSTEM DESCRIPTIONS

To more effectively compare the optimizers, a more complex suite of codes and multiple goal sets were required. A bit more complex than the sounding rocket, the single stage solid missile system design code[1] generates preliminary level engineering models of missiles powered by single stage solid propellant motors and flies them based on a set of 35 essential design parameters. The optimizers are used to manipulate these design parameters in order to develop missiles that achieve desired performance goals. This code has been used in multiple optimization studies[11,12,23] and has proven to be an accurate and reliable instrument for use in engineering applications.

After the program is initialized, it reads in various constants contained in two input files. These terms include mathematical constants, program limits, material properties, moments of inertia, component lengths and locations, and program goals. The entire component breakdown of the files is presented in Table 6. Some of these terms may be altered at some point within the code, but the variable arrays must be created with acceptable values at the start of the code.

Table 6. Component Breakdown of Missile Design Code Initial Constants

Component Section	Number of Variables
Constants	22
Material Densities	6
Program Lengths, Limits, and Constants	31
Constants and Set Numbers	25
Initiation of Launch Data	16
Target Data	6
GA Goals (outdata variables)	20
Auxiliary Variables to be Used as Needed	21
List of GA Variables Passed to Setup, etc.	35
Total Missile Variables	40
Guidance and Plotting Variables	29
Component Densities	30
Masses	30
Center of Gravity	30
Moments of Inertia	60
Component Lengths	30
Axial Starting Point of Components	30
Required and Computed Data for Aero	30
Other Dimensions	16
Internal Solid Rocket Grain Variables	14
Nozzle and Throat Variables	23
Other Computed Stage Variables	8

Once all of the constants are initialized, the program reads in the appropriate optimizer input file. Whether for one of the GA's or the RPSO, the file will contain the optimizer controlling parameters as well as the variables to be used in the optimization. Whenever a new member is created by the GA's or every time a particle moves in the RPSO, a completely new missile is created based on the initial constants and specified design parameter constraints. The missile flight is then simulated by a sequence of subroutines that determine the propulsion characteristics, mass properties, and aerodynamic properties. The performance and flight profile for the missile are determined by a 6-degree-of-freedom, or 6-DOF, routine.

For the 6-DOF to successfully fly the missile, all of its flight characteristics must be known. A number of parameters define the geometry and propellant characteristics for the missile, and from these parameters, the code is able to determine the thrust profile for the motor in a manner similar to that described in Section 4. The code uses the specified grain geometry to determine the part of the thrust equation that is independent of atmospheric conditions. The large volume left inside of the missile after the propellant has been burned required that the tail-off for the thrust be modeled in order to more accurately predict performance. Once the propulsion system is modeled, the mass properties for the missile are calculated and used to determine the center of gravity and moments of inertia for all component systems.

Before the 6-DOF routine can take control of the missile, its aerodynamic properties are determined using a fast predicting aerodynamic scheme called AERODSN[32]. AERODSN is very useful in that it is a non-linear, fast-predictive code, which is essential for use with the optimizers in order to provide a fairly high level of accuracy while simultaneously minimizing run times. However, this requires that certain assumptions be made primarily relating to missile geometry.

The 6-DOF simulates missile flight by integrating the equations of motion and using the previously calculated propulsion, physical, and aerodynamic characteristics in a single model. The program flies the missile in a spherical earth model using all six degrees of freedom to determine the missile’s overall performance and flight profile. The missile’s individual performance is then compared against the desired performance goals specified at the onset of the program, a fitness determined, and returned to the optimizer for analysis. A flow chart for the code is shown in Figure 10.

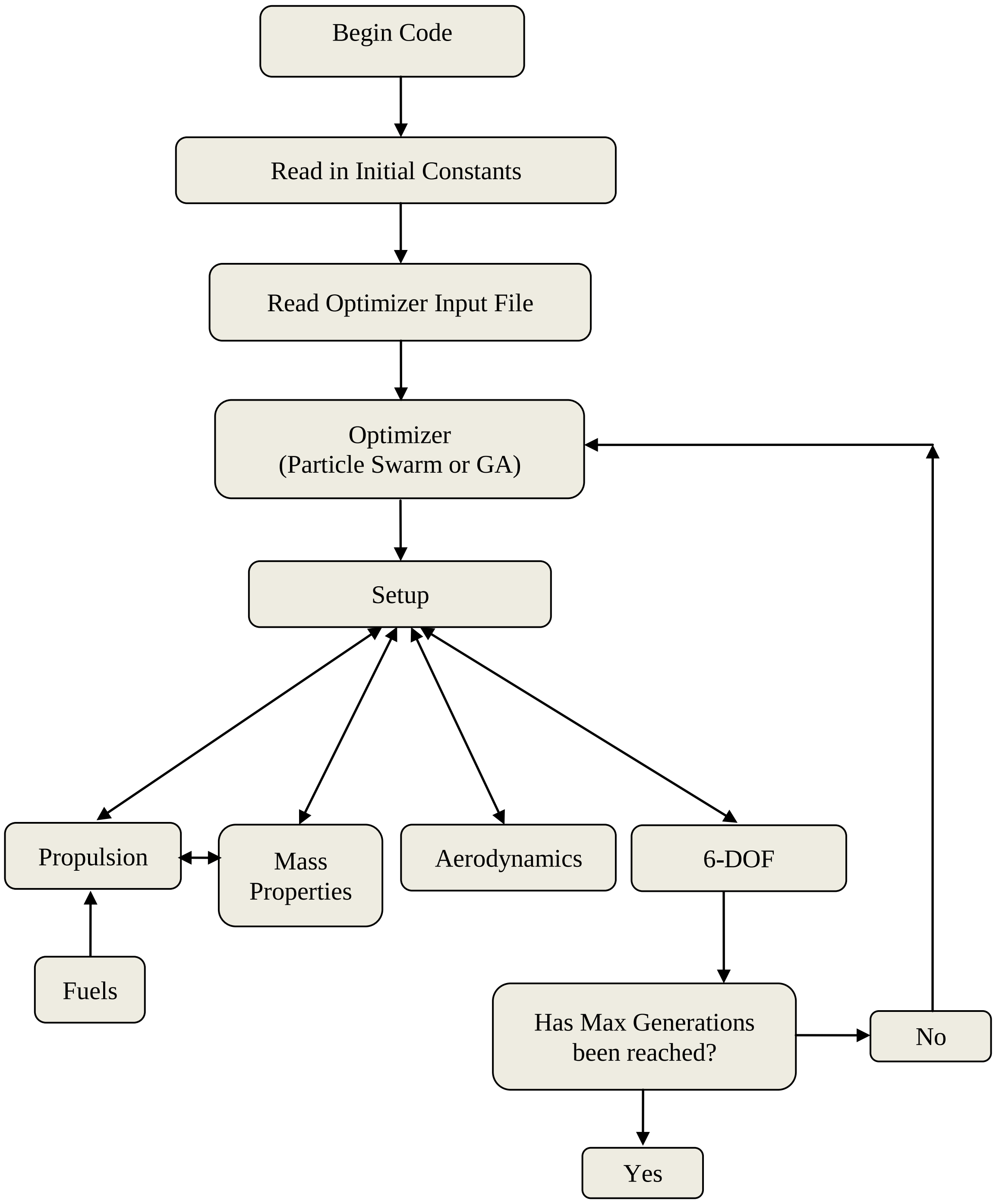

Begin Code
Read in Initial Constants
Read Optimizer Input File
Optimizer (Particle Swarm or GA)
Setup
Propulsion
Mass Properties
Aerodynamics
6-DOF
Fuels
Has Max Generations been reached?
No
Yes

Figure 10. Missile System Design Code Program Flow

For this study, a single stage solid motor was used in the design optimizations. In order to create a fully developed missile system, 35 design variables are used for the model. Table 7 lists the required variables for the model and separates them between the various types.

Table 7. Single Stage Solid Missile Variable Definitions

Missile Geometry	Propellant Properties	Autopilot Controls
Nose Radius Ratio = rnose/rbody	Fuel Type	Autopilot On Delay Time
Nose Length Ratio = lnose/dbody	Propellant Outer Radius Ratio rpvar=(rp+f)/rbody	Initial Launch Angle (deg)
Fractional Nozzle Length Ratio = f/ro	Propellant Inner Radius Ratio rivar=ri/rp	Pitch Multiplier Gain
Nozzle Throat Diameter/dbody	Number of star points	Yaw Multiplier Gain
Total Length of Stage1/dbody	Fillet Radius Ratio f/rp	Initial Elevator Angle (deg)
Diameter of Stage1 (dbody)	Epsilon - star width	Gainp2 – gain in pitch angle dif
Wing Exposed Semi-span = b2w/dbody	Star point angle	B2var = b2vane/rexit
Wing Root Chord = crw/dbody		Time Step to Actuate Controls
Wing Taper Ratio = ctw/crw		Gainy2 – gain in yaw angle dif
Leading Edge Sweep Angle Wing (deg)		Deltx for Z Corrections
xLEw/lbody		Deltx for Y Corrections
Tail Exposed Semi-span = b2t/dbody		
Tail Root Chord = Crt/dbody		
Tail Taper Ratio = ctt/crw		
Leading Edge Sweep Angle Tail (deg)		
xTEt/Lbody		
Nozzle Exit Dia/dbody		

As can be seen in the second column of Table 7, the solid motor grain design used in this system is very similar to that used in the sounding rocket program. The absent propellant property variables that are inherent to the sounding rocket are defined elsewhere in this program as it works to develop a more complete missile system. The third column of the table lists the variables used to control the autopilot for the program. These variables are used in case the missile is being guided by an active control system. This missile code possesses the ability to control the missile through tail fin deflections, nozzle vane deflections, or nozzle gimbaling. However, during this study, the autopilot system was turned off and the missiles were only allowed to fly ballistic trajectories. The first column of the table lists all of the parameters used to develop the external geometry of the missile including the nose, wings, tail fins, and nozzle[33]. The missile code designs a bell nozzle by fitting a parabola tangent to a circular arc throat section. The parabola is extended out until the required expansion ratio for the nozzle is met[33]. A more detailed description of the external geometry components of the missile can be seen in Reference 34.

8 SINGLE STAGE SOLID MISSILE SYSTEM OPTIMIZER COMPARISONS

All four of the optimization algorithms were compared using the single stage solid motor missile system code described in Section 7. To effectively compare the optimizers, they must be evaluated on multiple goals and goal sets. In the first set of optimizer tests, the two GA's, RPSO, and compound RPSO were used to match a specified range. By only requiring a match to one goal, the optimizers have a higher probability of finding a missile design that successfully meets the performance requirement. It would be possible for all four optimizers to find completely different, yet nearly equivalent optimal solutions. A more effective comparison of the optimizers, however, would require a more difficult goal set. That is why for the next set of tests, the optimizers were used to match various goal pairs. By matching two separate goals, such as range and takeoff weight, the probability of finding good solutions significantly decreases. The design parameters for all four of the optimizers are constrained by the same maximum and minimum allowable values so as to allow for the optimizers to potentially choose the same or similar designs. The input files for the optimizers can be seen in Appendices F – H.

8.1 Single Stage Solid - Match Range 250,000 ft

For the first optimizer test, the algorithms were required to match a single goal: design a missile that has a range of 250,000 ft. For all of these tests, the optimizers were allowed to perform 100,000 function evaluations and once again, were compared on overall fitness and on speed of convergence. Due to the lower amount of function evaluations allowed for these tests and its overall performance in the sounding rocket test, the hybrid RPSO was run using 5 particles with a 5% search area for the mini-swarm. The fitness for matching range is calculated as follows:

$$Fitness = \frac{abs(Range - DesiredRange)}{10} \tag{20}$$

The corresponding convergence histories for the optimizers are shown in Figure 11.

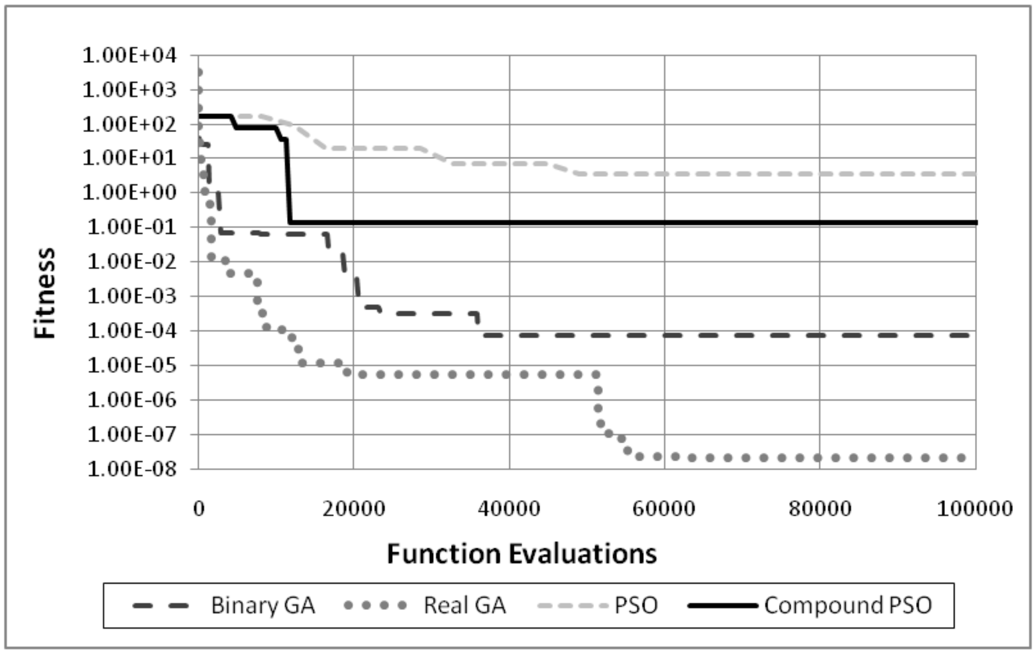


Figure 11. Single Stage Solid Match Range 250,000ft Convergence History

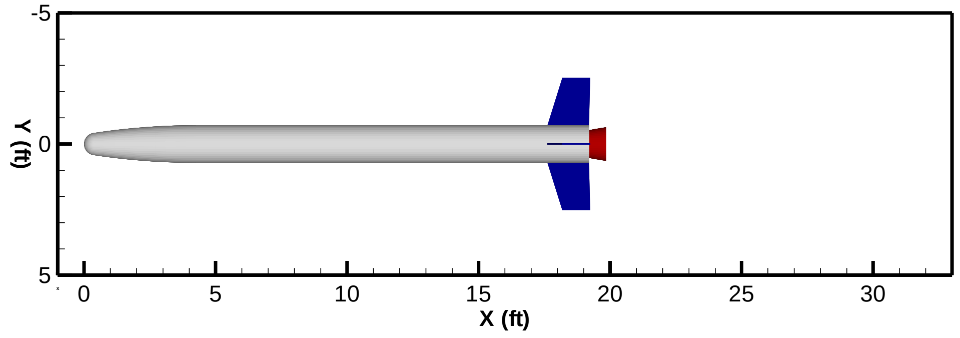
The real-coded GA was able to achieve the best fitness over the entire length of the optimization run. It was able to maintain a fitness nearly an order of magnitude better than the binary GA, which was the second best overall performer. Neither the conventional RPSO nor the compound RPSO were able to outperform either of the GA's. The compound RPSO, however, was able to outperform the standard RPSO in both speed of convergence and overall fitness value. The scale of the graph should be noted. Even though the compound particle swarm did not match the desired range as closely as the GA's, it still missed the target by less than 5 ft, which might be considered a sufficient answer.

As the previous particle swarm input parameter study showed, the velocity parameters for the swarm can have a profound effect on the optimizer's performance. Similar results could possibly be demonstrated for the parameters of the mini-swarm. A varied set of control parameters could result in increased solution convergence for the compound RPSO.

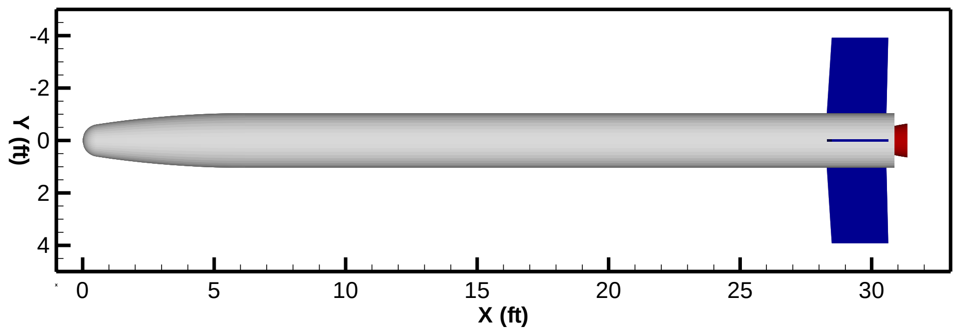
The final missile design parameters and relative 3-D missile plots are shown in Table 8 and Figure 12, respectively. There are significant differences between all four of the optimizer designs. Interestingly, the compound RPSO design matches much more closely to the two GA designs than it does the standard RPSO. This optimizer test could have shown how single goal problems have the ability to produce completely separate but nearly equally optimal results, yet all of the optimizers appeared to be converging on a similar solution. There seems to be a direct correlation to the size of the missile and the fitness value. The real GA was obviously able to match the goal much more closely, but the compound RPSO still only missed the target range by less than one missile diameter.

Table 8. Single Stage Solid 250,000ft Range Design Parameters

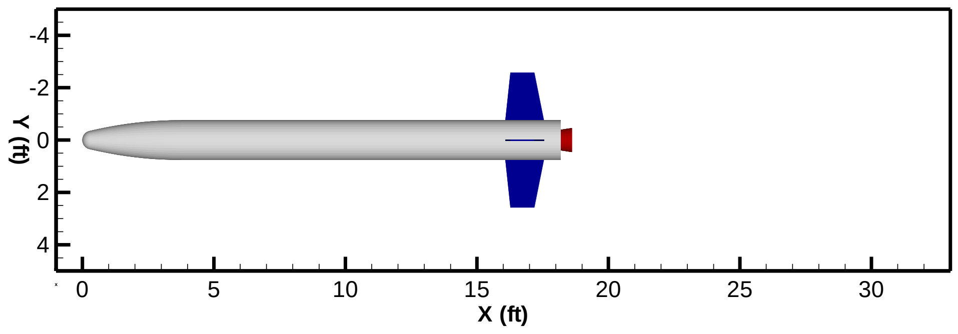
PARAMETER	COMPOUND RPSO	RPSO	BINARY GA	REAL GA
rnose/rbody	0.5940	0.5940	0.4667	0.4921
lnose/dbody	2.9700	2.9700	2.4677	2.1681
fuel type	7.7000	8.9100	4.2000	3.6886
star out R (rpvar)	0.6410	0.6167	0.5857	0.4981
star inner ratio	0.6043	0.7920	0.3800	0.1907
number of star pts	7.7251	10.8900	10.6000	7.5305
fillet radius ratio	0.0468	0.0990	0.0953	0.0565
eps	0.6700	0.9405	0.8333	0.8321
star point angle (deg)	1.2394	7.3168	6.4000	2.2710
fractional nozzle length	0.8517	0.6784	0.8171	0.9489
dia throat/dbody	0.2983	0.2970	0.2730	0.2794
fineness ratio	13.3776	14.8500	12.0000	12.2367
dia stage 1 (m)	0.4375	0.6336	0.4622	0.3952
wing semispan/dbody	0.0362	0.0495	0.0271	0.0415
wing root chord/dbody	0.0361	0.0495	0.0214	0.0303
wing taper ratio	0.9125	0.9468	0.9300	0.9830
wing LE sweep angle (deg)	5.3544	13.3342	7.4444	1.1327
xLEw/lbody	0.2330	0.2475	0.2143	0.2423
tail semispan/dbody	1.2577	1.3860	1.2000	1.3276
tail root chord/dbody	1.0925	1.0890	0.9667	1.0370
tail taper ratio	0.6744	0.9483	0.6196	0.8493
tail LE sweep anlge (deg)	17.2962	3.7229	6.0635	22.5363
xTEt/lbody	0.9994	0.9900	0.9643	0.9686
autopilot delay time (sec)	5000.0000	5000.0000	5000.0000	4999.1740
initial launch angle (deg)	65.5544	73.7197	67.8571	67.2771
pitch multiplier gain	3.7484	3.9793	3.9200	4.1139
yaw multiplier gain	1.7744	1.1841	2.3889	1.1935
nozzle exit dia/dbody	0.8933	0.6196	0.6033	0.8466
initial pitch cmd angle (deg)	-7.0000	-7.0000	-11.2667	-11.4603
gain in pitch	0.0035	0.0043	0.0100	0.0042
b2var=b2vane/rexit	0.0050	0.0090	0.0000	0.0085
time step to actuate noz (sec)	0.4301	0.9307	0.4400	0.9656
gain in yaw	0.0055	0.0024	0.0100	0.0075
deltx corrections for z	0.2712	0.7056	0.0000	0.3523
deltx corrections for y	0.9599	0.6725	0.0000	0.2061
FITNESS	**0.142455**	**3.704469**	**7.41948E-05**	**2.24332E-08**



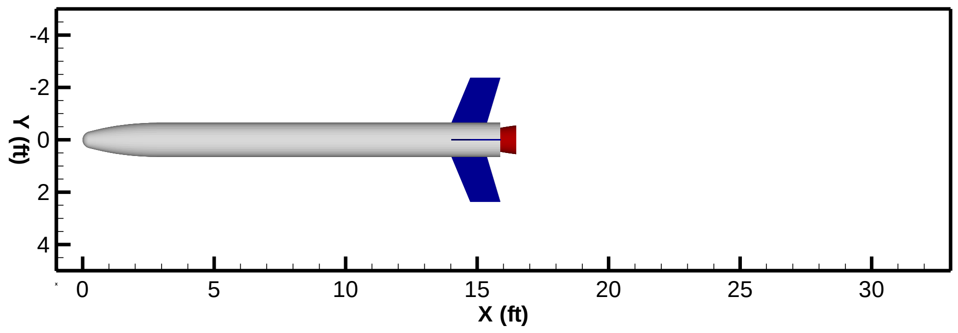
Compound RPSO



RPSO



Binary GA



Real GA

Figure 12. Single Stage Solid 250,000ft Range 3-D Models

8.2 Single Stage Solid - Match Range 750,000 ft

For the second optimizer test, the algorithms were once again required to match a range goal. This time they were expected to design a missile that has a range of 750,000 ft. Although similar to the previous test, this evaluation is still valuable because it helps to further evaluate the optimizers by forcing them to explore a separate part of the solution space. Once again, the optimizers were allowed to perform 100,000 function evaluations and were compared on overall fitness and on how fast they converged. The corresponding convergence histories for the optimizers are shown in Figure 13.

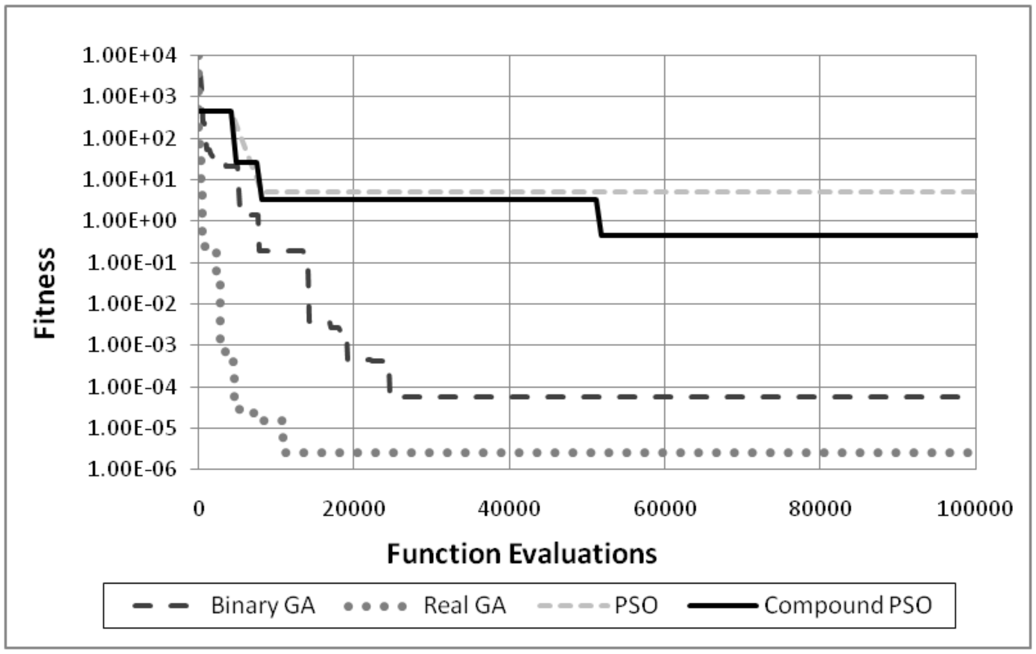


Figure 13. Single Stage Solid Match Range 750,000ft Convergence History

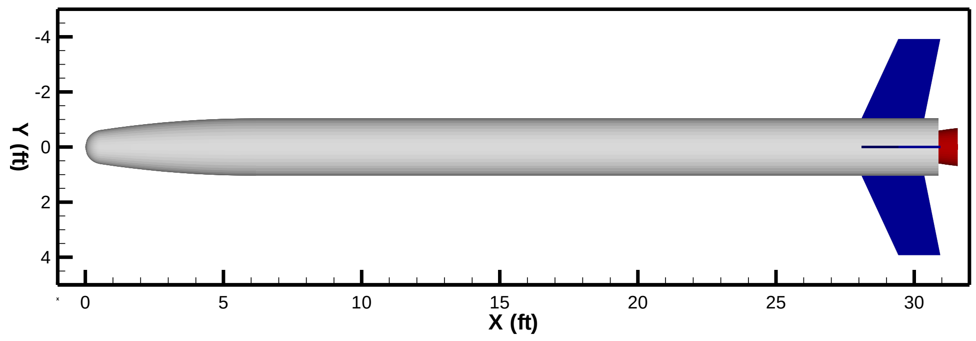
Just as in the previous match range problem, the real-coded GA completely outperformed the other optimizers in both convergence speed and overall fitness. The RPSO's also failed once again to outperform either of the GA's. The RPSO was unable to make any further improvements after the initial fitness jump that occurred after only a tenth of the allowable function evaluations were performed. The compound RPSO was able to overcome this and produce a design with a fitness value an order of magnitude lower.

The final missile design parameters and relative 3-D missile plots are shown in Table 9 and Figure 14, respectively. As with the previous optimizer test, four completely different designs were developed by the optimizers. Interestingly, the two RPSO designs are more similar to each other, while at the same time, the two GA designs are more similar to each other. This could be a testament to the differences between the social behavior theory of the Particle Swarms and the evolution theory of the Genetic Algorithms.

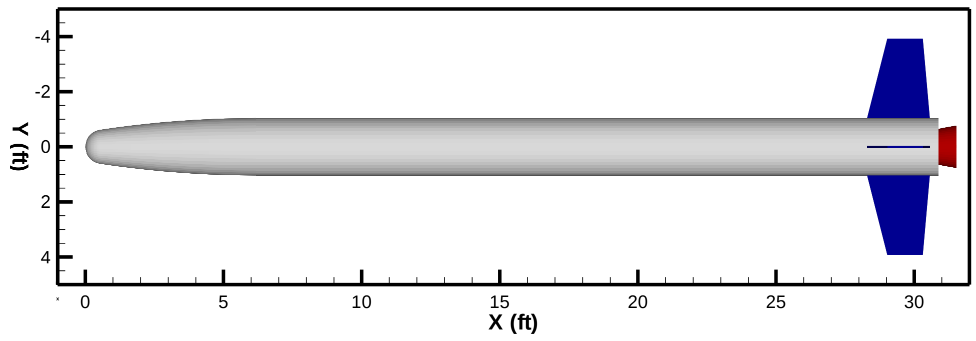
The design chosen by the compound RPSO happened to be one of the larger missiles again, but the size disparity between it and the GA's designs happened to be much larger than in the previous test. The hybrid RPSO optimizer design ended up missing the target by less than 5 ft. This seems like a fairly good result considered it is only just over twice the missile diameter and less than $5.8x10^{-4}$% error. The two GA's achieved this level of accuracy after less than 20,000 function evaluations.

Table 9. Single Stage Solid 750,000ft Range Design Parameters

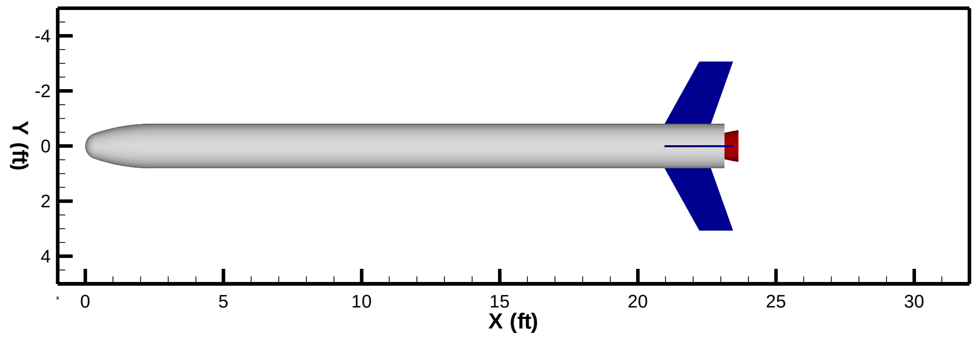
PARAMETER	COMPOUND RPSO	RPSO	BINARY GA	REAL GA
rnose/rbody	0.5940	0.5940	0.6000	0.4686
lnose/dbody	2.9700	2.9700	1.5000	2.4161
fuel type	3.0716	7.1135	6.3333	4.8264
star out R (rpvar)	0.7920	0.7920	0.4429	0.5630
star inner ratio	0.1965	0.5741	0.3333	0.2903
number of star pts	10.4201	10.8900	10.2000	9.3306
fillet radius ratio	0.0990	0.0990	0.0533	0.0608
eps	0.9405	0.9405	0.7944	0.7660
star point angle (deg)	6.9040	9.3931	5.2000	2.3099
fractional nozzle length	0.9801	0.7402	0.6862	0.8173
dia throat/dbody	0.2970	0.2970	0.2563	0.2679
fineness ratio	14.8500	14.8500	14.3333	12.5506
dia stage 1 (m)	0.6336	0.6336	0.4919	0.5579
wing semispan/dbody	0.0495	0.0495	0.0500	0.0373
wing root chord/dbody	0.0495	0.0495	0.0386	0.0231
wing taper ratio	0.9801	0.9126	0.9540	0.9874
wing LE sweep angle (deg)	29.7000	8.2556	3.3016	5.1092
xLEw/lbody	0.2475	0.2475	0.2071	0.2321
tail semispan/dbody	1.3860	1.3860	1.4000	1.3079
tail root chord/dbody	1.0890	1.0890	1.0333	1.0480
tail taper ratio	0.6678	0.5651	0.7276	0.6418
tail LE sweep anlge (deg)	24.7908	14.2276	29.0794	13.9887
xTEt/lbody	0.9836	0.9900	0.9786	0.9903
autopilot delay time (sec)	5000.0000	5000.0000	5000.0000	4999.3909
initial launch angle (deg)	46.9515	57.1010	61.4286	65.3084
pitch multiplier gain	4.3312	3.6348	3.9200	4.0316
yaw multiplier gain	1.2869	1.0999	2.0714	1.7802
nozzle exit dia/dbody	0.6604	0.7374	0.7100	0.6705
initial pitch cmd angle (deg)	-7.0000	-7.0000	-9.1333	-12.9696
gain in pitch	0.0055	0.0046	0.0100	0.0082
b2var=b2vane/rexit	0.0022	0.0099	0.0000	0.0057
time step to actuate noz (sec)	0.5286	0.3815	0.9067	0.3822
gain in yaw	0.0037	0.0079	0.0000	0.0068
deltx corrections for z	0.3883	0.2938	1.0000	0.7611
deltx corrections for y	0.5040	0.1351	0.0000	0.1520
FITNESS	**0.43482**	**5.04715**	**5.76720E-05**	**2.48252E-06**



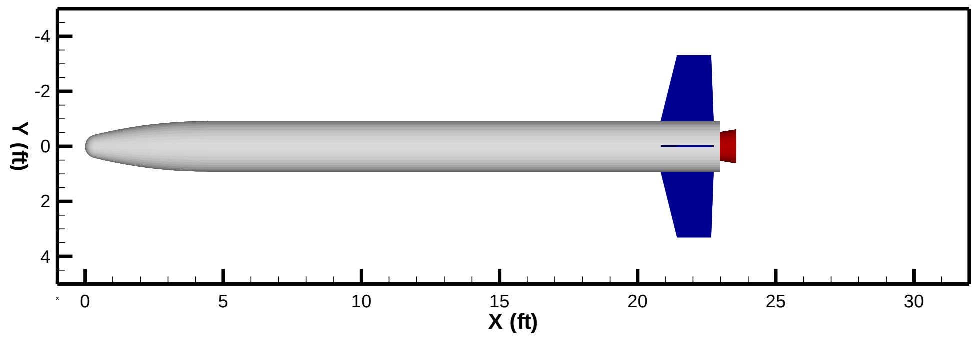
Compound RPSO



RPSO



Binary GA



Real GA

Figure 14. Single Stage Solid 750,000ft Range 3-D Models

8.3 Single Stage Solid - Match Range 100,000 ft Takeoff Weight 2,500 lb

To more thoroughly test the optimizers, three more complex goal sets were chosen for comparison. The first goal set required the algorithms to match a range of 100,000 ft while also matching a takeoff weight of 2,500 lb. As with the single goal tests, the optimizers were allowed to perform 100,000 function evaluations. Because the optimizers are required to match two goals, they are driven more towards a global solution that minimizes both goals. The optimization only required the minimization of single fitness value as before. The fitness is determined by summing the goals as follows:

$$Fitness = \frac{abs(Range - DesiredRange)}{10} + \frac{abs(Weight - DesiredWeight)}{10} \tag{21}$$

The corresponding convergence histories for the optimizers are shown in Figure 15.

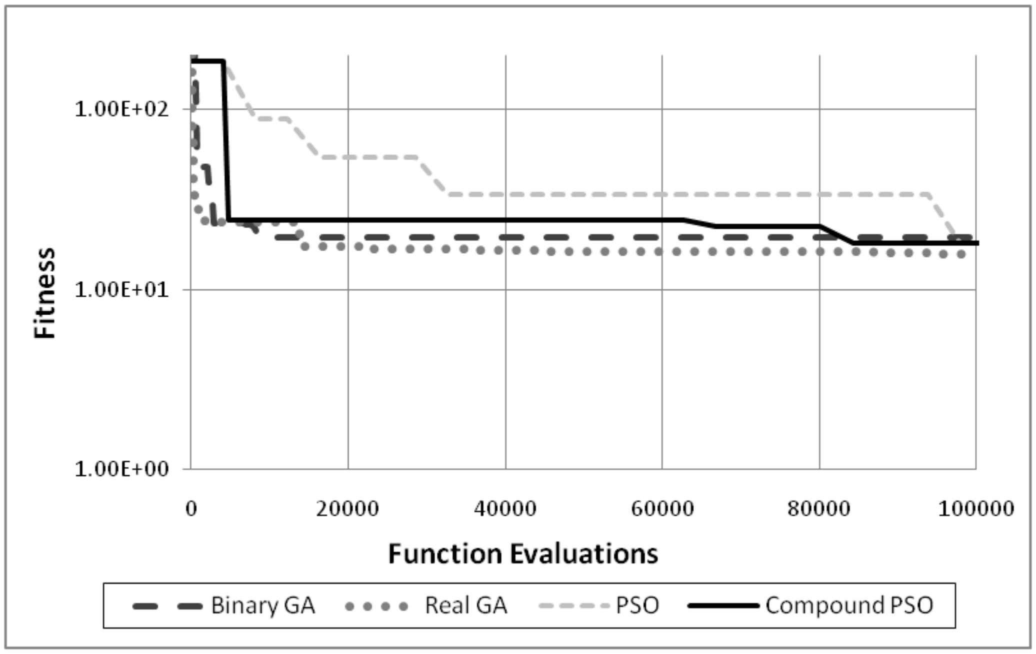


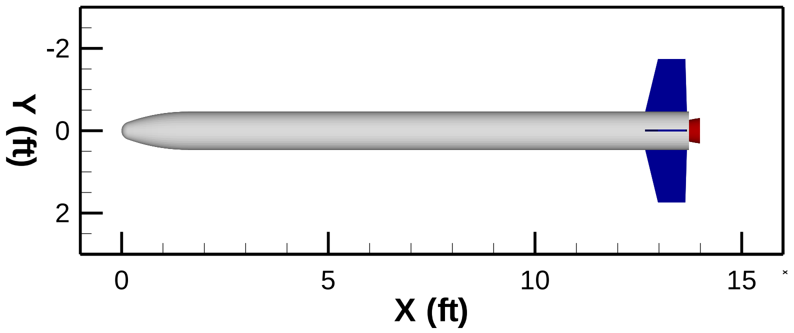
Figure 15. Single Stage Solid Match Range 100,000ft Weight 2,500lb Convergence History

Though still the best performer, the real-coded GA did not perform overwhelmingly better than the other optimizers. The compound RPSO outperformed the standard particle swarm for the entire optimization run, with the standard RPSO only coming close with the last large fitness jump. This final jump by the hybrid particle swarm after 80,000 function evaluations actually pushed it to a better fitness than the binary GA. As previously speculated, the optimizers all appear to be driven toward a global minimization solution. There is very little difference between the four fitness values. Because the fitness values are relatively high, the actual optimal solution to this minimization problem may not have been defined within the prescribed set of allowable parameters. The indication of these results is that the compound RPSO is a competitive algorithm for complex problems.

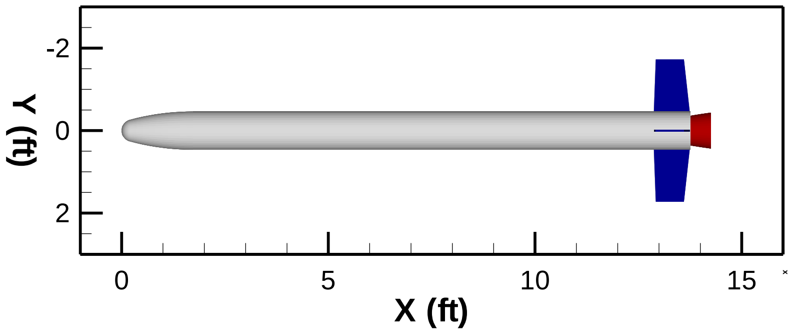
Table 10 and Figure 16 show the final missile design parameters and relative 3-D missile plots, respectively. Unlike the previous tests, there does not appear to be as much diversity between the designs developed by the optimizers. As expected, the worst performer, the binary GA, ended up being most unlike the other designs. The graphical representation for the convergence for the RPSO's is somewhat skewed compared to the GA's. The current best fitness values for all of the optimizers are recorded only at the end of each generation/iteration. For the real GA, this is after every function evaluation, and for the binary GA, this is after every 200 function evaluations. The RPSO's iterations are much longer, and the best fitness values are reported only after more than 4,000 function evaluations. This accounts for a much less gradual graphical representation of convergence for the RPSO's. The compound RPSO may appear more gradual due to the fitness value being reported after the much shorter mini-swarm iterations.

Table 10. Single Stage Solid 100,000ft Range 2,500lb Weight Design Parameters

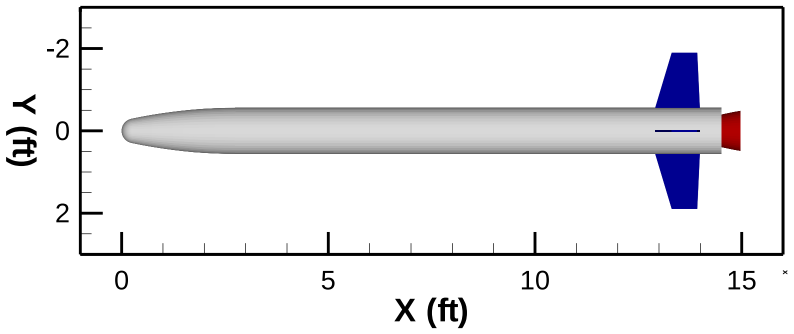
PARAMETER	COMPOUND RPSO	RPSO	BINARY GA	REAL GA
rnose/rbody	0.4787	0.5869	0.5333	0.4150
lnose/dbody	1.7441	1.8614	2.4677	2.4533
fuel type	6.5340	2.5608	6.3333	7.3769
star out R (rpvar)	0.5462	0.5870	0.7286	0.7065
star inner ratio	0.3410	0.1010	0.5200	0.3949
number of star pts	6.5239	10.5705	8.6000	9.4862
fillet radius ratio	0.0532	0.0760	0.0813	0.0484
eps	0.8666	0.8236	0.8889	0.8546
star point angle (deg)	3.9955	6.3574	7.6000	5.4533
fractional nozzle length	0.7367	0.9801	0.7386	0.9640
dia throat/dbody	0.2751	0.2970	0.2524	0.2534
fineness ratio	14.8500	14.8500	13.0000	12.5510
dia stage 1 (m)	0.2816	0.2823	0.3401	0.3332
wing semispan/dbody	0.0322	0.0185	0.0214	0.0209
wing root chord/dbody	0.0472	0.0455	0.0386	0.0415
wing taper ratio	0.9452	0.9504	0.9240	0.9228
wing LE sweep angle (deg)	22.9509	1.0712	11.5873	16.8577
xLEw/lbody	0.2274	0.2475	0.2500	0.2392
tail semispan/dbody	1.3860	1.3642	1.2000	1.2000
tail root chord/dbody	1.0890	0.9325	0.9667	0.9674
tail taper ratio	0.6547	0.7858	0.5694	0.9270
tail LE sweep anlge (deg)	13.5621	2.1120	16.6508	9.0799
xTEt/lbody	0.9965	0.9990	0.9643	0.9769
autopilot delay time (sec)	5000.0000	5000.0000	5000.0000	4999.1318
initial launch angle (deg)	54.1655	57.4390	41.4286	57.1196
pitch multiplier gain	4.2039	4.0119	3.9200	4.3300
yaw multiplier gain	1.0298	2.5035	2.4286	1.7933
nozzle exit dia/dbody	0.6646	0.9398	0.8700	0.9483
initial pitch cmd angle (deg)	-7.0000	-7.0000	-8.0667	-11.9299
gain in pitch	0.0001	0.0084	0.0100	0.0044
b2var=b2vane/rexit	0.0069	0.0000	0.0100	0.0043
time step to actuate noz (sec)	0.4129	0.3977	0.7200	0.7346
gain in yaw	0.0085	0.0036	0.0100	0.0076
deltx corrections for z	0.2438	0.6875	0.0000	0.1884
deltx corrections for y	0.8862	0.7156	0.0000	0.8768
TOTAL FITNESS	**18.04673**	**18.19579**	**19.63227**	**15.82746**
Range Error	**0.64565**	**5.52699**	**0.85501**	**0.04880**
Weight Error	**17.40109**	**12.66880**	**18.77725**	**15.77866**



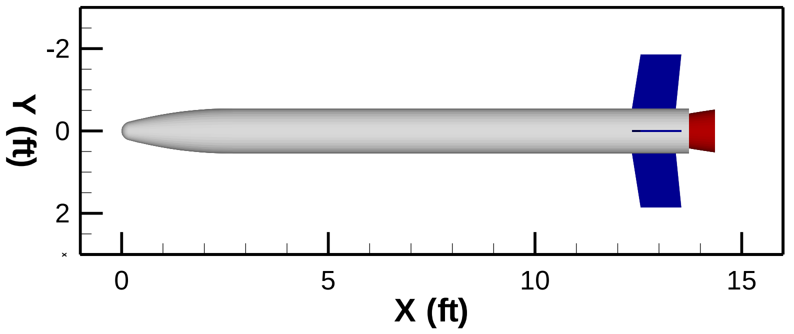
Compound RPSO



RPSO



Binary GA



Real GA

Figure 16. Single Stage Solid 100,000ft Range 2,500lb Weight 3-D Models

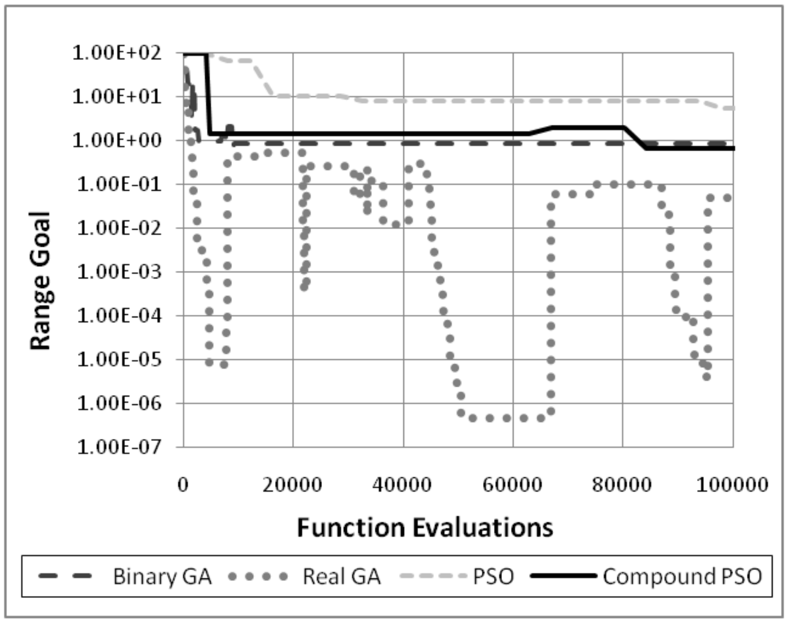


Figure 17. Match Range 100,000ft Error, Test 3

Figure 17 shows the solution convergence plot for the match range goal of the optimization. Only the real GA had large fluctuations in its range error while the RPSO's and binary GA maintained nearly constant values after their initial improvements, with only small changes during the rest of the run. Figure 18 shows the convergence history for the weight error.

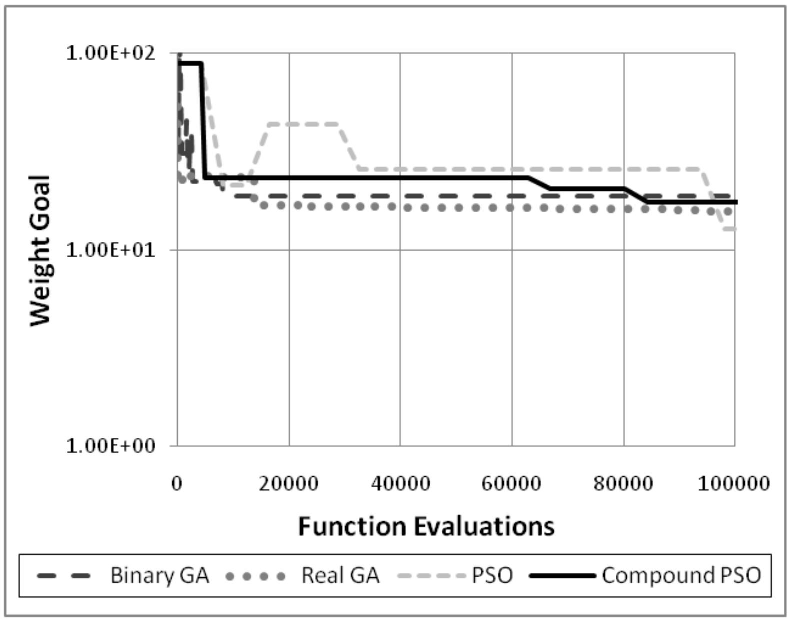


Figure 18. Match Weight 2,500lb Error, Test 3

For this goal, only the standard RPSO showed large fluctuations in its weight error, while the other optimizers remained mostly constant with only small improvements for the majority of the run. The standard RPSO managed to be the worst performer in the range goal and the best performer in the weight goal. Interestingly, the compound RPSO's improvement over the binary GA involved it having better answers for both the range goal and the weight goal.

8.4 Single Stage Solid - Match Range 350,000 ft Takeoff Weight 4,000 lb

The second two goal test required the optimizers to match a range of 350,000 ft while also matching a takeoff weight of 4,000 lb. Figure 19 shows the convergence histories for the optimizers.

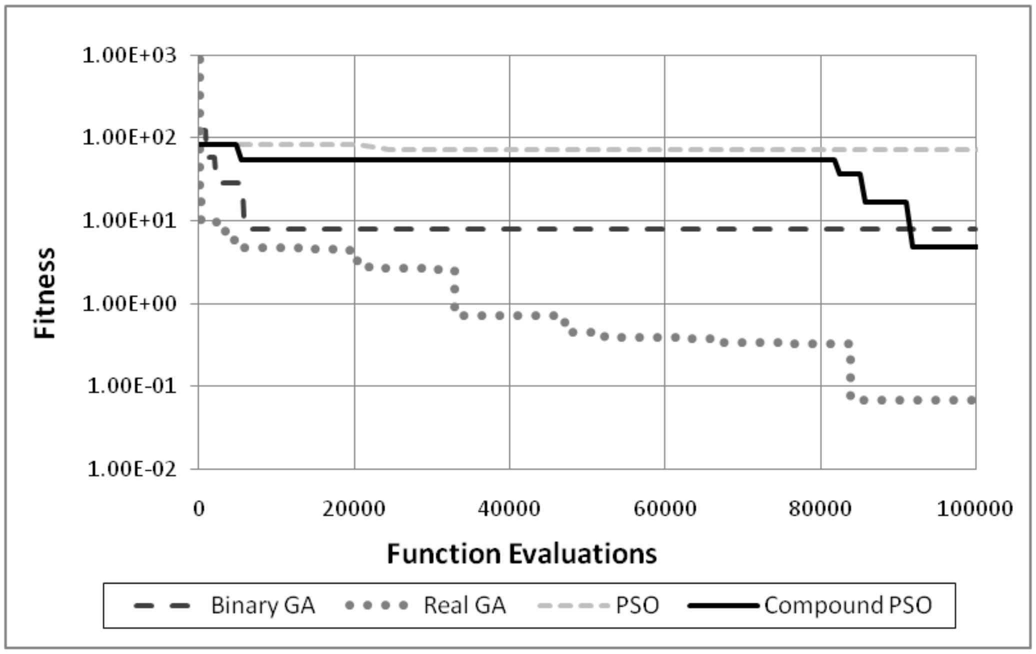


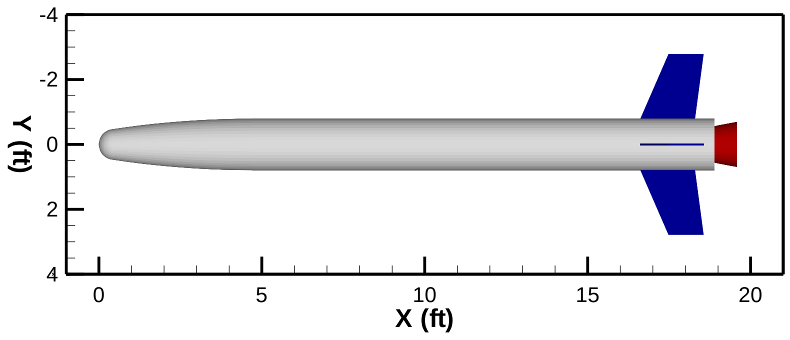
Figure 19. Single Stage Solid Match Range 350,000ft Weight 4,000lb Convergence History

Once again, the real GA ended up being the best performing optimizer. The compound RPSO was able to outperform the single phase particle swarm optimizer for this goal and produce an answer better than the binary GA. The hybrid RPSO's ability to take advantage of a much more thorough local search is very apparent around the 80,000 function evaluation mark. The standard RPSO was unable to make any significant improvements after its initial jump, but the hybrid was able to make some significant improvements with the use of the mini-swarm. The optimizers were not grouped as closely together as with the previous range-weight goal set and the run produced multiple orders of magnitude improvement between the four fitness values.

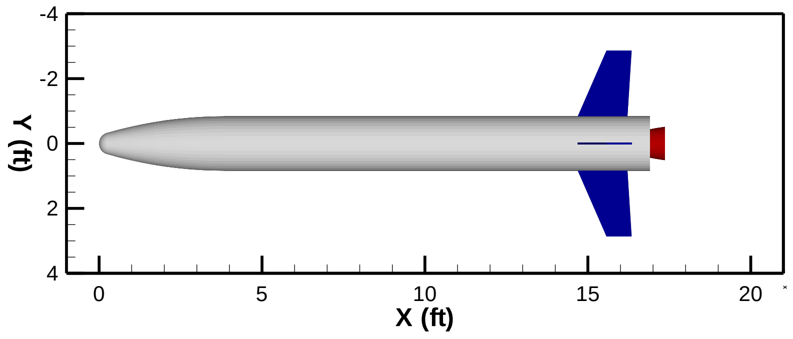
Table 11 and Figure 20 show the final missile design parameters and relative 3-D missile plots, respectively. Looking at these, the compound RPSO effectiveness becomes more apparent. The compound RPSO was able to develop a unique design compared to the three other optimizers, and although not as accurate as the real-coded GA, its fitness is better than that of the binary GA. This test run shows how making multiple calls to the mini-swarm after failing to improve can prove effective. The standard RPSO was unable to improve over the length of the run, but because the hybrid continued to call upon the mini-swarm, it was able to make a very significant improvement before reaching the maximum number of function evaluations.

Table 11. Single Stage Solid 350,000ft Range 4,000lb Weight Design Parameters

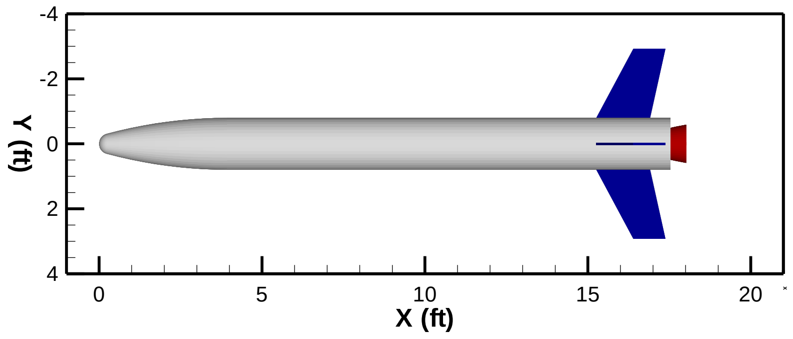
PARAMETER	COMPOUND RPSO	RPSO	BINARY GA	REAL GA
rnose/rbody	0.5940	0.4040	0.4000	0.5450
lnose/dbody	2.9700	2.2931	2.4677	2.6525
fuel type	7.3291	3.8625	2.6000	5.3379
star out R (rpvar)	0.5016	0.5360	0.5142	0.4674
star inner ratio	0.3103	0.2878	0.7533	0.3902
number of star pts	9.1741	5.0500	9.4000	5.9667
fillet radius ratio	0.0877	0.0733	0.0720	0.0636
eps	0.6408	0.6060	0.8611	0.7068
star point angle (deg)	4.8039	6.2506	2.2000	7.7415
fractional nozzle length	0.7767	0.7463	0.6914	0.7006
dia throat/dbody	0.2559	0.2548	0.2865	0.2661
fineness ratio	11.9358	10.1000	11.0000	11.4019
dia stage 1 (m)	0.4824	0.5099	0.4858	0.4766
wing semispan/dbody	0.0246	0.0344	0.0214	0.0265
wing root chord/dbody	0.0233	0.0450	0.0214	0.0405
wing taper ratio	0.9143	0.9090	0.9480	0.9719
wing LE sweep angle (deg)	27.2583	27.8624	12.5079	20.4673
xLEw/lbody	0.2053	0.2020	0.2285	0.2206
tail semispan/dbody	1.2577	1.2120	1.3333	1.3692
tail root chord/dbody	1.0566	0.9090	1.0333	0.9740
tail taper ratio	0.6420	0.5050	0.5964	0.5899
tail LE sweep anlge (deg)	23.3743	23.4372	28.1587	12.5368
xTEt/lbody	0.9682	0.9595	0.9642	0.9867
autopilot delay time (sec)	5000.0000	5000.0000	4999.0000	4999.2974
initial launch angle (deg)	75.7407	64.9833	53.5714	70.8039
pitch multiplier gain	3.8389	3.6360	4.2933	3.6346
yaw multiplier gain	2.6228	1.3101	3.0634	1.7781
nozzle exit dia/dbody	0.8719	0.6135	0.7366	0.7861
initial pitch cmd angle (deg)	-7.0000	-7.0000	-11.2666	-9.8070
gain in pitch	0.0100	0.0053	0.0000	0.0030
b2var=b2vane/rexit	0.0078	0.0000	0.0000	0.0062
time step to actuate noz (sec)	0.8941	0.3030	0.8600	0.4220
gain in yaw	0.0017	0.0003	0.0100	0.0068
deltx corrections for z	0.0607	0.7626	0.0000	0.3531
deltx corrections for y	0.0382	0.5597	1.0000	0.6857
TOTAL FITNESS	**4.76180**	**71.77821**	**8.05037**	**0.06887**
Range Error	**2.65134**	**71.63351**	**0.76689**	**0.05249**
Weight Error	**2.11046**	**0.14468**	**7.28348**	**0.01638**



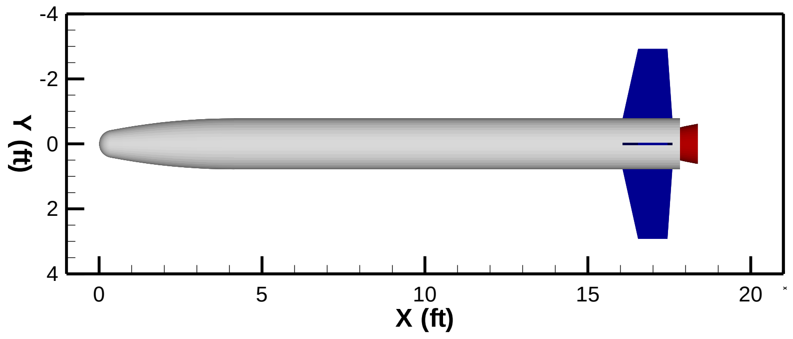
Compound RPSO



RPSO



Binary GA



Real GA

Figure 20. Single Stage Solid 350,000ft Range 4,000lb Weight 3-D Models

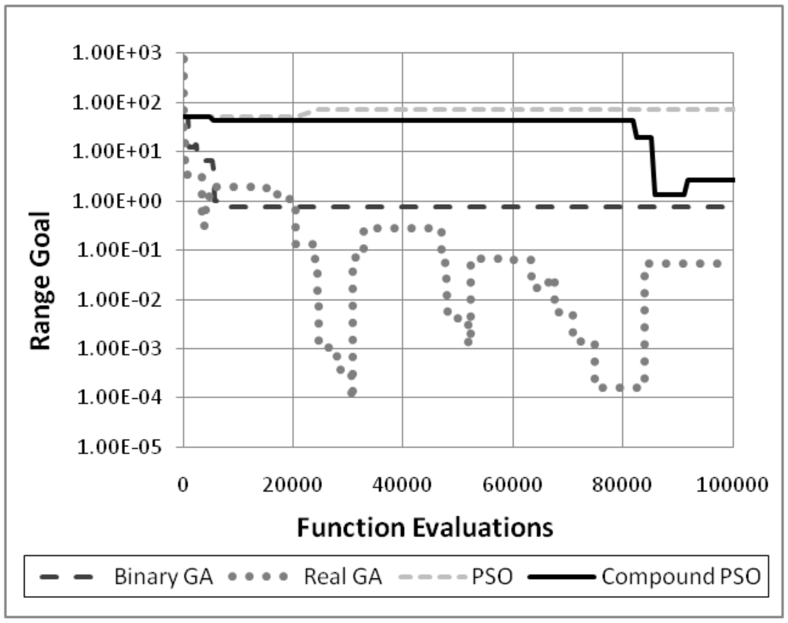


Figure 21. Match Range 350,000ft Error, Test 4

Figure 21 shows the solution convergence plot for the range error. The real-coded GA once again had large fluctuations in the range error while the other optimizers only had a small number of changes in the range error. The compound RPSO maintained a poor range error close to that of the standard RPSO until the final call of the mini-swarm managed to provide a drastically better solution. Figure 22 shows the weight error convergence history for the optimizers.

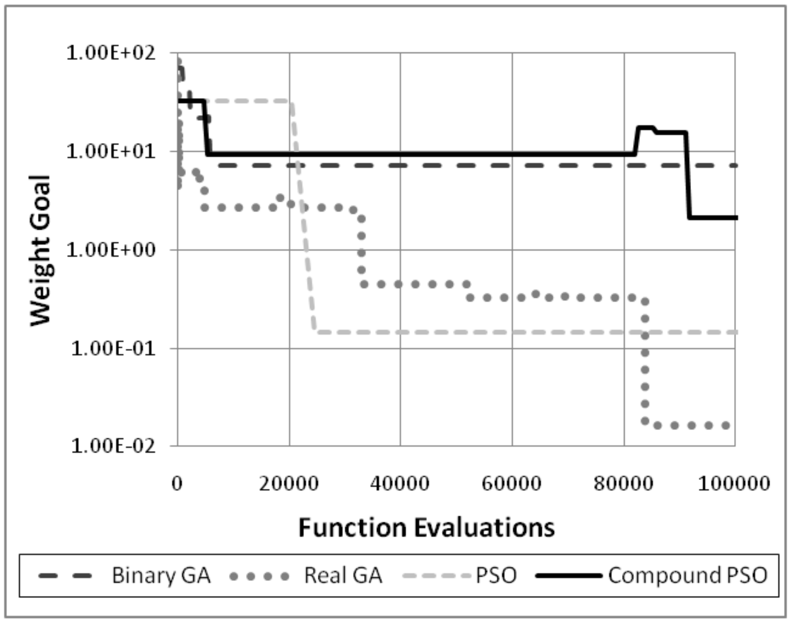


Figure 22. Match Weight 4,000lb Error, Test 4

The real GA continued to improve steadily with only minor fluctuations, while the binary GA showed no changes after about 7,000 function evaluations. The mini-swarm of the compound RPSO managed to make significant improvements in both goal errors at the end of the run. The hybrid RPSO was actually able to match the weight much more closely than the binary GA, accounting for its overall better fitness. The standard RPSO was able to effectively minimize the weight goal, but its inability to similarly minimize the range goal ultimately led to its poor performance.

8.5 Single Stage Solid - Match Time of Flight 240 sec Takeoff Weight 2,500 lb

The final two goal test required the optimizers to match a 240 sec time of flight while also matching a takeoff weight of 2,500 lb. The optimizers have already proven their ability to match range goals, so this test uses another performance parameter for evaluation. Once again, the optimization only required the minimization of a single fitness value as before. The fitness is determined by summing the goals as follows:

$$Fitness = \frac{abs(TOF - DesiredTOF)}{10} + \frac{abs(Weight - DesiredWeight)}{10} \tag{22}$$

Figure 23 shows the convergence histories for this optimization test.

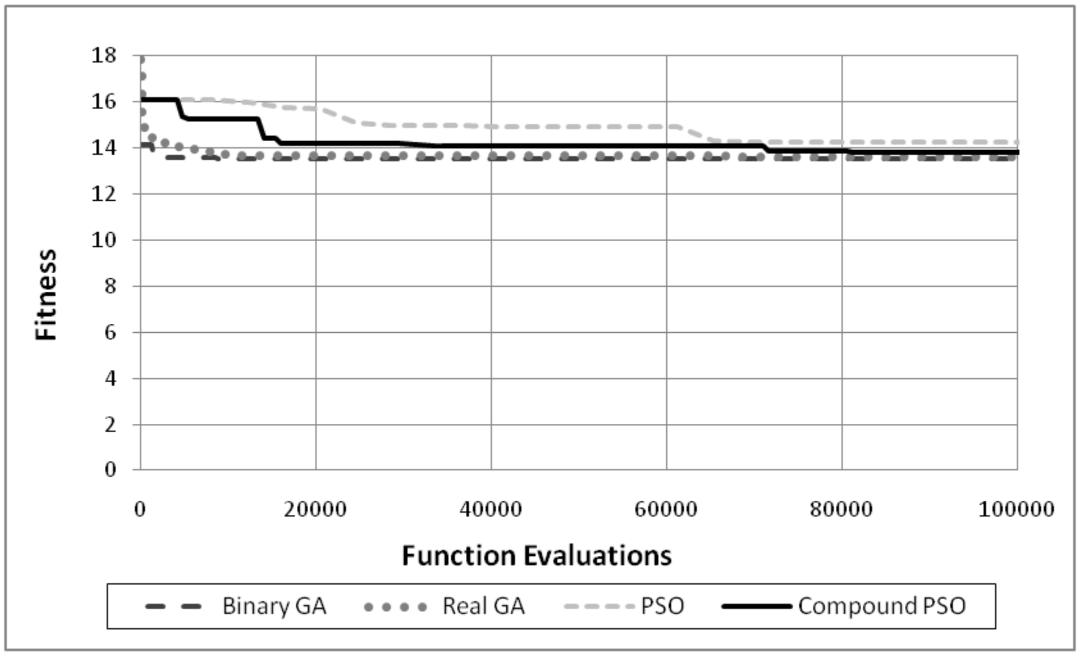


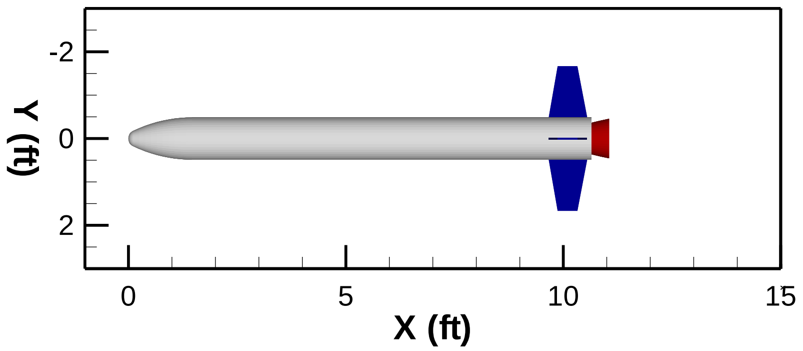
Figure 23. Single Stage Solid Match TOF 240sec Weight 2,500lb Convergence History

The binary GA ended up being the best performer of this optimization test, but did not perform overwhelmingly better than the other optimizers. The compound RPSO outperformed the standard particle swarm for most of the optimization run and was able to develop an answer very near that of the GA's. The optimizers were once again driven toward a global minimization solution with there being very little difference between the four fitness values. The relatively high fitness values may have resulted from the actual optimal solution not being defined within the prescribed set of allowable parameters.

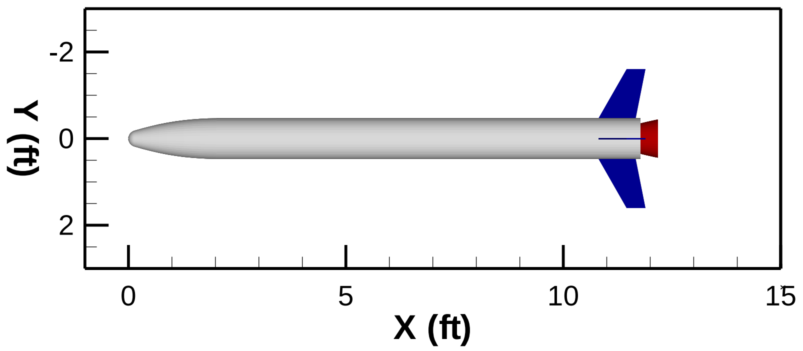
The final missile design parameters and relative 3-D missile plots are shown in Table 12 and Figure 24, respectively. The hybrid RPSO was once again able to develop a design completely unlike the others while maintaining a performance very comparable to the GA's. Other than the nose parameters, most of the other design parameters for the hybrid are significantly different than those of the other optimizers.

Table 12. Single Stage Solid 240sec TOF 2,500lb Weight Design Parameters

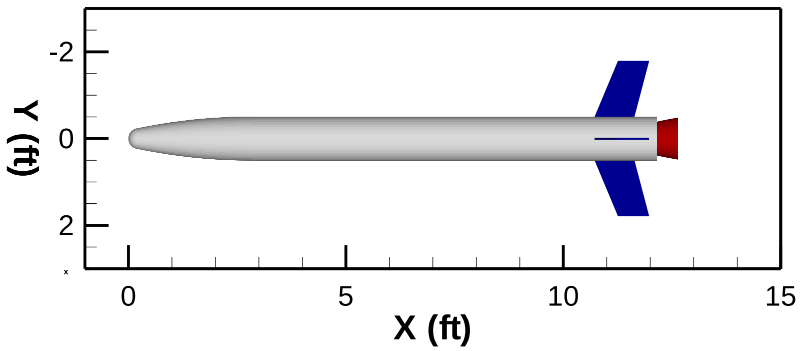
PARAMETER	COMPOUND RPSO	RPSO	BINARY GA	REAL GA
rnose/rbody	0.4040	0.4040	0.4667	0.4000
lnose/dbody	1.5048	2.1651	2.8065	1.6679
fuel type	2.9327	5.1677	2.0667	2.5834
star out R (rpvar)	0.5231	0.5159	0.5857	0.7075
star inner ratio	0.2099	0.2807	0.2400	0.5570
number of star pts	5.0439	5.1944	11.0000	10.3661
fillet radius ratio	0.0888	0.0601	0.1000	0.0468
eps	0.6135	0.6419	0.7889	0.7175
star point angle (deg)	4.9593	6.5429	5.8000	3.5387
fractional nozzle length	0.6704	0.6781	0.7752	0.9069
dia throat/dbody	0.2532	0.2525	0.2627	0.2500
fineness ratio	10.9351	12.5823	12.0000	11.5040
dia stage 1 (m)	0.2967	0.2853	0.3088	0.3180
wing semispan/dbody	0.0110	0.0467	0.0157	0.0104
wing root chord/dbody	0.0348	0.0495	0.0157	0.0473
wing taper ratio	0.9087	0.9090	0.9120	0.9399
wing LE sweep angle (deg)	2.7909	12.7349	18.0317	18.8796
xLEw/lbody	0.2021	0.2020	0.2357	0.2064
tail semispan/dbody	1.2120	1.2120	1.2667	1.2000
tail root chord/dbody	0.9059	0.9090	0.9000	0.9440
tail taper ratio	0.5114	0.5050	0.7817	0.7500
tail LE sweep anlge (deg)	9.9175	29.7000	22.6349	30.0000
xTEt/lbody	0.9907	0.9900	0.9571	0.9683
autopilot delay time (sec)	5000.0000	5000.0000	5000.0000	4999.5958
initial launch angle (deg)	83.7936	84.1500	85.0000	85.0000
pitch multiplier gain	4.3675	4.3560	3.9733	3.6810
yaw multiplier gain	3.4244	3.4650	3.4206	1.2194
nozzle exit dia/dbody	0.9355	0.9405	0.9500	0.9483
initial pitch cmd angle (deg)	-7.0000	-7.0000	-9.6667	-7.9205
gain in pitch	0.0099	0.0099	0.0100	0.0052
b2var=b2vane/rexit	0.0088	0.0099	0.0000	0.0049
time step to actuate noz (sec)	0.9950	0.9900	0.3467	0.5563
gain in yaw	0.0097	0.0099	0.0000	0.0065
deltx corrections for z	0.9910	0.9900	1.0000	0.5178
deltx corrections for y	0.9929	0.9900	1.0000	0.2823
TOTAL FITNESS	**13.79352**	**14.23533**	**13.53377**	**13.59890**
TOF Error	**13.79294**	**13.68900**	**13.53333**	**13.59890**
Weight Error	**0.00058**	**0.54633**	**0.00045**	**7.27596E-13**



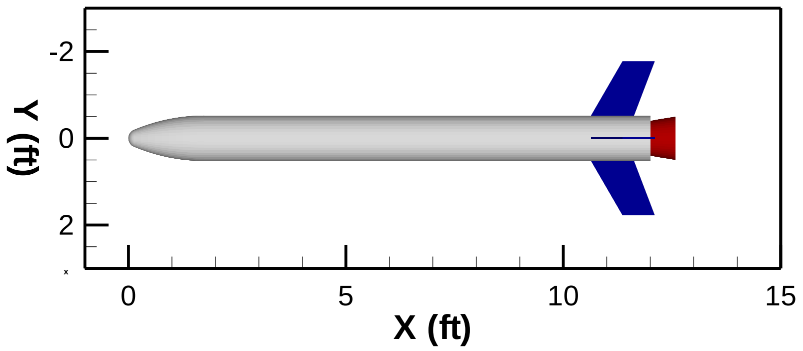
Compound RPSO



RPSO



Binary GA



Real GA

Figure 24. Single Stage Solid 240sec TOF 2,500lb Weight 3-D Models

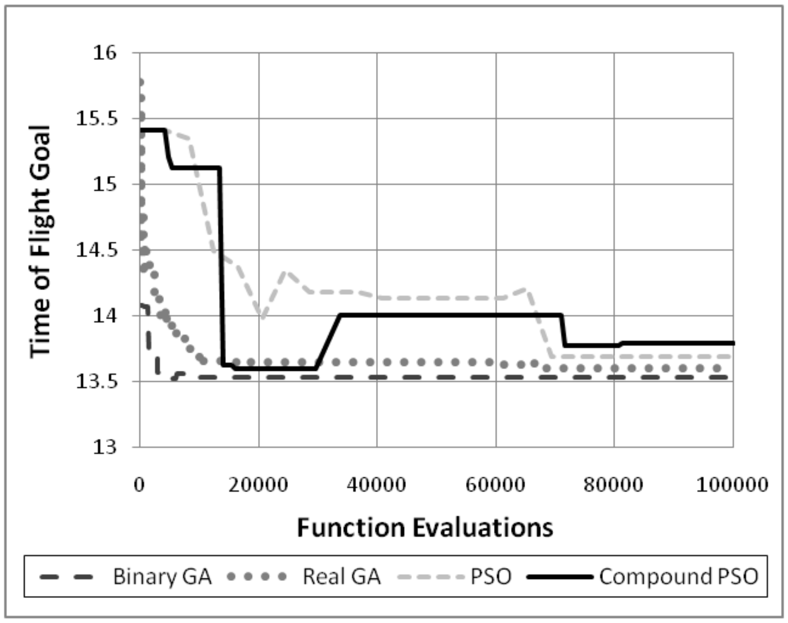


Figure 25. Match TOF 240sec Error, Test 5

Figure 25 shows the time of flight error history for the optimizers. The error for this goal accounted for the majority of the error for all of the optimizers. It appears the solution space might have limited the time of flight error to around 13.5. Figure 26 shows the convergence history for the weight errors.

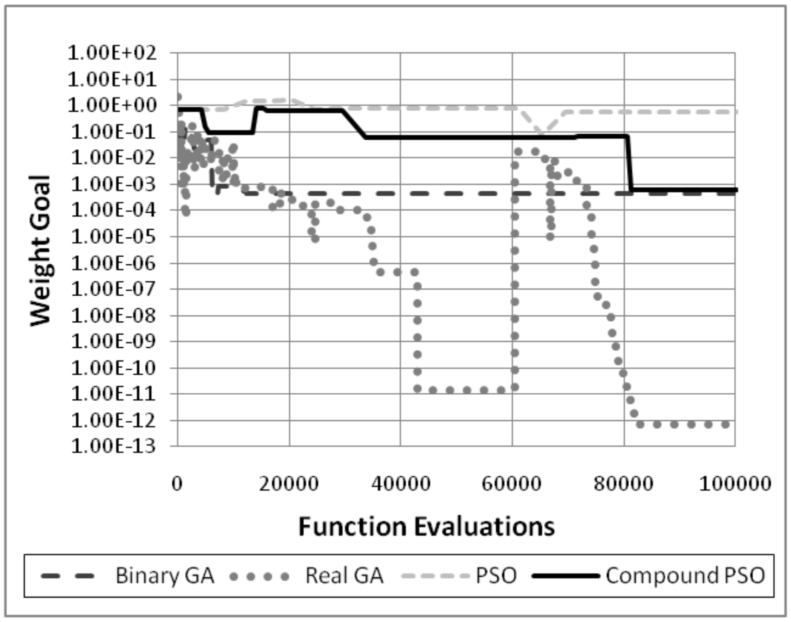


Figure 26. Match Weight 2,500lb Error, Test 5

All of the optimizers performed fairly well at matching the weight goal for this problem. Other than the single-phase RPSO, the optimizers were able to match the weight within a hundredth of a pound. The added accuracy of the real GA is purely academic and may make the graph a bit misleading. Because the optimizers were able to match the weight so well and the time of flight so poorly, this most likely means it would require a much larger missile to achieve the required flight time. The optimizers probably chose to match the weight goal instead of the time goal because the weight goal is an order of magnitude larger and resulted in much higher fitness values when time of flight was matched. This discrepancy could be addressed by goal weighting.

Table 13 shows the overall fitness results for all four of the optimizers for the five single stage solid missile test cases performed.

Table 13. Final Optimizer Fitness Comparisons

	COMPOUND RPSO	RPSO	BINARY GA	REAL GA
Match 250,000ft	0.142455	3.704469	7.41948E-05	2.24332E-08
Match 750,000ft	0.434820	5.047148	5.76720E-05	2.48252E-06
Match 100,000ft 2,500lb	18.046730	18.19579	19.63227	15.82746
Match 350,000ft 4,000lb	4.761799	71.778210	8.05037	0.06887
Match 240sec 2,500lb	13.793520	14.235330	13.53377	13.59890

9 CONCLUSIONS AND RECOMMENDATIONS

A repulsive particle swarm optimizer has been developed by compounding a RPSO for use on complex engineering problems. This compound RPSO was developed based on a previously modified version of the standard RPSO that was adapted to make the algorithm compatible with reasonable engineering problems. To demonstrate the utility of this approach, this new optimization algorithm has been applied to a complex rocket propulsion application. The standard RPSO algorithm was first coupled to a solid motor sounding rocket design code so that a study could be performed on its input parameters and how they affect optimizer performance. From these results, the compound particle swarm optimizer was developed by imbedding a second, smaller particle swarm within the original algorithm. These two forms of the RPSO were then attached to a full 6-DOF single stage solid missile design code and compared against a binary and real-coded GA. A number of single and two goal tests were performed in which the optimizers were allowed to run for 100,000 function evaluations in order to compare their fitness convergence. Even in its early stages of development, the compound particle swarm optimizer showed significant potential as a useful engineering tool when compared to the genetic algorithms. The hybrid RPSO was generally able to converge much faster than the standard RPSO, but did not necessarily always have the best fitness at the end of the optimization run. Although unable to outperform the GA's, the particle swarm was shown to be effective. The compound RPSO actually seemed to be more competitive on the more complex problems.

Further studies on the compound RPSO’s mini-swarm control parameters are recommended to determine whether a more optimum set exists for better overall convergence or to determine varying sets of input parameters for use on different problems. It may also be beneficial to use a separate optimizer to optimize the input parameters for use on certain problems or for different engineering applications. The standard RPSO’s inability to continuously improve on some of the goals may show that the optimized input parameters found in the sounding rocket study may not be the best parameters for use on the full missile code. The hybrid swarm also showed its ability to make improvements when the single-phase could not by making repeated calls to the mini-swarm after the larger swarm failed to show improvement. Changing the frequency of these calls may also have a profound impact on the overall solution.

The compound RPSO proved extremely useful on cases when the standard RPSO would fail to improve during most of the run. Studies have also shown that the number of particles, the percentage of the solution space, and the number of iterations allowed for the mini-swarm can have a significant impact on the solution convergence. A fundamental modification to the local search portion of the particle swarm optimizer may make it more efficient and may lead to dramatically improved results. For example, the Pattern Search algorithm may be used on the best performer at the completion of the mini-swarm runs in order to improve results. The RPSO and especially the compound RPSO are relatively new tools for use in this area, but the preliminary results have proven promising. The compound RPSO was even able to outperform the binary GA on two of the multi-goal test cases. However, the RPSO’s are still not able to converge to solutions nearly as quickly as the GA’s. The current compound RPSO algorithm has shown a vast improvement to Mishra’s particle swarm algorithm with plenty of room left for further advancement.

REFERENCES

[1]J.E. Burkhalter, R.M. Jenkins, and R.J. Hartfield, M. B. Anderson, G.A. Sanders, “Missile Systems Design Optimization Using Genetic Algorithms,” AIAA Paper 2002-5173, Classified Missile Systems Conference, Monterey, CA, November, 2002.

[2]Karr, C.L., Freeman, L.M., and Meredith, D.L., "Genetic Algorithm based Fuzzy Control of Spacecraft Autonomous Rendezvous," NASA Marshall Space Flight Center, Fifth Conference on Artificial Intelligence for Space Applications, 1990.

[3]Krishnakumar, K., Goldberg, D.E., “Control System Optimization Using Genetic Algorithms”, Journal of Guidance, Control, and Dynamics, Vol. 15, No. 3, May-June 1992.

[4]Torella, G., Blasi, L., “The Optimization of Gas Turbine Engine Design by Genetic Algorithms”, AIAA Paper 2000-3710, 36th AIAA/ASME/SAE/ASEE Joint Propulsion Conference and Exhibit, July 2000.

[5]Perhinschi, M.,G., "A Modified Genetic Algorithm for the Design of Autonomous Helicopter Control System," AIAA-97-3630, Presented at the AIAA Guidance, Navigation, and Control Conference, New Orleans, LA, August 1997.

[6]Mondoloni, S., “A Genetic Algorithm for Determining Optimal Flight Trajectories”, AIAA Paper 98-4476, AIAA Guidance, Navigation, and Control Conference and Exhibit, August 1998.

[7]Anderson, M.B., “Using Pareto Genetic Algorithms for Preliminary Subsonic Wing Design”, AIAA Paper 96-4023, presented at the 6th AIAA/NASA/USAF Multidisciplinary Analysis and Optimization Symposium, Bellevue, WA, September 1996.

[8]Perez, R.E., Chung, J., Behdinan, K., “Aircraft Conceptual Design Using Genetic Algorithms”, AIAA Paper 2000-4938, Presented at the 8th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, September 2000.

[9]Anderson, M.B., Burkhalter, J.E., and Jenkins, R.M., "Design of an Air to Air Interceptor Using Genetic Algorithms", AIAA Paper 99-4081, presented at the 1999 AIAA Guidance, Navigation, and Control Conference, Portland, OR, August 1999.

[10]Anderson, M.B., Burkhalter, J.E., and Jenkins, R.M., “Intelligent Systems Approach to Designing an Interceptor to Defeat Highly Maneuverable Targets”, AIAA Paper 2001-1123, presented at the 39th Aerospace Sciences Meeting and Exhibit, Reno, NV, January 2001.

[11]Hartfield, Roy J., Jenkins, Rhonald M., Burkhalter, John E., “Ramjet Powered Missile Design Using a Genetic Algorithm,” AIAA 2004-0451, presented at the forty-second AIAA Aerospace Sciences Meeting, Reno NV, January 5-8, 2004.

[12]Jenkins, Rhonald M., Hartfield, Roy J., and Burkhalter, John E., **“**Optimizing a Solid Rocket Motor Boosted Ramjet Powered Missile Using a Genetic Algorithm”, AIAA 2005-3507 presented at the Forty First AIAA/ASME/SAE/ASEE Joint Propulsion Conference, Tucson, AZ, July 10-13, 2005.

[13]Burger, Christoph and Hartfield, Roy J., “Propeller Performance Optimization using Vortex Lattice Theory and a Genetic Algorithm”, AIAA-2006-1067, presented at the Forty-Fourth Aerospace Sciences Meeting and Exhibit, Reno, NV, Jan 9-12, 2006.

[14]Chernyavsky, B., Stepanov, V., Rasheed, K., Blaize, M., and Knight, D., “3-D Hypersonic Inlet Optimization Using a Genetic Algorithm”, AIAA Paper 98-3582, 34th AIAA/ASME/SAE/ASEE Joint Propulsion Conference and Exhibit, July 1998.

[15]Hu, X., Eberhart, R., and Shi, Y., “Engineering Optimization with Particle Swarm”, IEEE Swarm Intelligence Symposium, Indianapolis, IN, 2003.

[16]Vukovic, S., Sopta, L., “Binary Coded and Real Coded Genetic Algorithm in Pipeline Flow”, AMS Paper 35-42, 7 th International Applied Mathematical Communications Conference, 1998.

[17]Elliot, L., Ingham, D. B., Kyne, K.G., “A Real Coded Genetic Algorithm for the Optimization of Reaction Rate Parameters for Chemical Kinetic Modeling in a Perfectly Stirred Reactor”, Proceeding of Genetic and Evolutionary Computational Conference, New York, 2002.

[18]Sugala, S. V., Bhattacharaya, P.K., “Real Coded Genetic Algorithm for Optimization of Pervaporation Process Parameters for Removal of Volatile Organics from Water”, Ind. Eng. Chem. Res. 2003, Volume 42, No.13, 3118-3128, November 2003.

[19]Holland, *Adaptation in Natural and Artificial Systems*, The MIT Press; Reprint edition 1992 (originally published in 1975).

[20]Mishra, S. K., “Global Optimization by Particle Swarm Method: A Fortran Program”, Munich Personal RePEc Archive, Munich University, Munich, DE, 2006 (unpublished).

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

[22]Anderson, M.,B., “Users Manual for IMPROVE© Version 3.0”, Sverdrup Technology Inc./TEAS Group.

[23]Dyer, John D., Hartfield, Roy J., Dozier, Gerry V., and Burkhalter, John E., “Aerospace Design Optimization Using a Steady State Real-Coded Genetic Algorithm”, AIAA Paper 2008-5921, 12th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, September 2008.

[24]Eshelman, L.J., Schaffer, J.D., “Foundations of Genetic Algorithms 2”, Real Coded Genetic Algorithms and Interval Schemata, 5-17, San Mateo, CA, 1992.

[25]Vavak, F., Fogarty, T. *Comparison of Steady State and Generational Genetic Algorithms for Use in Non-Stationary Environments*, IEEE Press, 1996.

[26]Dorigo, Marco, Montes de Oca, Marco A., and Engelbrecht, Andries, (2008), "Particle Swarm Optimization", *Scholarpedia,* 3(11):1486.

[27]Badyrka, Jeffrey M., Jenkins, Rhonald M., and Hartfield, Roy J., "Aerospace Design: A Comparative Study of Optimizers", AIAA Paper 2010-1311, 48th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition, January 2010.

[28]Burkhalter, J., Personal Communication, 22 February 2011.

[29]Hartfield, R., Jenkins, R., Burkhalter, J., and Foster, W., "A Review of Analytical Methods for Solid Rocket Motor Grain Analysis", *Journal of Spacecraft and Rockets,* Vol. 41 No.4, July-August 2004, pp.689-693.

[30]Hartfield, R., (2009), "Solid Rocket Motors: AERO 5530/6530/6536", [Lecture notes], Auburn, AL: Auburn University, Aerospace Engineering Department.

[31]Jenkins, Rhonald M., and Hartfield, Roy J., "Hybrid Particle Swarm-Pattern Search Optimizer for Aerospace Propulsion Applications", 46th AIAA/ASME/SAE/ASEE Joint Propulsion Conference & Exhibit, July 2010.

[32]Sanders, G.A. and Washington, W.D., "Computer Program for Estimating Stability Derivatives of Missile Configurations - Users Manual", U.S. Army Missile Command, August 1982.

[33]Huzel, D. K., and Huang, D. H., "Modern Engineering for Design of Liquid-Propellant Rocket Engines", Vol. 147, American Institute of Aeronautics and Astronautics, Washington DC, 1992.

[34]Burkhalter, John E., Jenkins, Rhonald M. and Hartfield, Roy J., "Genetic Algorithms for Missile Analysis," Final Report, Missile and Space Intelligence Center, Redstone Arsenal, AL, February 2003.

APPENDIX A: Sounding Rocket Binary GA Input File

```
.false.                          ; micro
.false.                          ; pareto
.false.                           ; steady_state
.false.                          ; maximize
.true.                           ; elitist
.true.                           ; creep
.false.                          ; uniform
.false.                          ; restart
.true.                           ; remove_dup
.false.                          ; niche
.false.                          ; phenotype
0.04                             ; niche diversity percentile goal
61742                            ; iseed
0.9                              ; pcross
0.002                            ; pmutation
0.05                             ; pcreep
3                                ; ngoals
1.,1.,1.                         ; xgls(j)
1.                               ; domst
2550                             ; convrg_chk (end of group2)
11                               ; no_para

 'kfuel 1'     ,  8.0   , 1.0   , 1.0   , .false.      ;xmax,xmin,resolution,niche_par
 'rpvar 2'     ,  0.95  , 0.1   , 0.01  , .false.      ;xmax,xmin,resolution,niche_par
 'rivar 3'     ,  0.99  , 0.01  , 0.01  , .false.      ;xmax,xmin,resolution,niche_par
 'nsp 4'       ,  17.0  , 3.0   , 2.0   , .false.      ;xmax,xmin,resolution,niche_par
 'fvar 5'      ,  0.2   , 0.01  , 0.01  , .false.      ;xmax,xmin,resolution,niche_par
 'eps 6'       ,  0.9   , 0.1   , 0.1   , .false.      ;xmax,xmin,resolution,niche_par
 'ptang 7'     ,  10.0  , 1.0   , 0.1   , .false.      ;xmax,xmin,resolution,niche_par
 'gl  8'       ,  800.  , 100.  , 2.5   , .false.      ;xmax,xmin,resolution,niche_par
 'rbi  9'      ,  24.   , 2.0   , 0.5   , .false.      ;xmax,xmin,resolution,niche_par
 'diath 10'    ,  20.0  , 0.5   , 0.1   , .false.      ;xmax,xmin,resolution,niche_par
 'ratio'       ,  10.   , 1.5   , 0.1   , .false.      ;xmax,xmin,resolution,niche_par
   1                               ; ifreq
   400                             ; mempops
   625                             ; maxgen
```

APPENDIX B: Sounding Rocket Real GA Input File

```
.false.                     ;generational  ONLY 1 GA TYPE CAN BE
.true.                      ;steady_state  ONLY 1 GA TYPE CAN BE
.false.                     ;hybrid        ONLY 1 GA TYPE CAN BE
.false.                     ;uniform x (50% parent1 and parent2)
.true.                      ;Blend x (blend of parents)
.false.                     ;singlepointx
.false.                     ;var_mutation  true allows mutation
2000                        ;kcheck # of gen before 1/5 rule ck
0.2                         ;xmutation_rate  how much mutation
.1                          ;xmutation_amount % of variables mutate
3.                          ;ngoals
1.,1.,1.                    ;xgls(j)
11                          ;no_para

'kfuel 1'    ,  8.0   , 1.0   , 1.0   , .false.     ;xmax,xmin,resolution,niche_par
'rpvar 2'    ,  0.95  , 0.1   , 0.01  , .false.     ;xmax,xmin,resolution,niche_par
'rivar 3'    ,  0.99  , 0.01  , 0.01  , .false.     ;xmax,xmin,resolution,niche_par
'nsp 4'      ,  17.0  , 3.0   , 2.0   , .false.     ;xmax,xmin,resolution,niche_par
'fvar 5'     ,  0.2   , 0.01  , 0.01  , .false.     ;xmax,xmin,resolution,niche_par
'eps 6'      ,  0.9   , 0.1   , 0.1   , .false.     ;xmax,xmin,resolution,niche_par
'ptang 7'    ,  10.0  , 1.0   , 0.1   , .false.     ;xmax,xmin,resolution,niche_par
'gl  8'      ,  800.  , 100.  , 2.5   , .false.     ;xmax,xmin,resolution,niche_par
'rbi  9'     ,  24.   , 2.0   , 0.5   , .false.     ;xmax,xmin,resolution,niche_par
'diath 10'   ,  20.0  , 0.5   , 0.1   , .false.     ;xmax,xmin,resolution,niche_par
'ratio'      ,  10.   , 1.5   , 0.1   , .false.     ;xmax,xmin,resolution,niche_par
1                           ; ifreq
30                          ; mempops
250000                      ; maxgen
```

APPENDIX C: Sounding Rocket RPSO Input File

```
0               ;maximize objective function = 1, minimize objective function = 0
30              ;population size, n
10              ;neighboring population sample size, nn (must be less than n)
100             ;maximum allowable number of independent variables, mx
5               ;local sample space size, nstep (5 < nstep <15)
100             ;results displayed every "nprn" iteration, nprn
1               ;chaos parameter, nsigma (=0 no chaotic perturbation, =1 chaotic perturbation)
3               ;neighborhood topology type, itop (=1 ring, =2 ring and random, =3 random)
0.75d0          ;particle velocity term 1 constant, a1
0.05d0          ;particle velocity term 2 constant, a2
0.25d0          ;particle velocity term 3 constant, a3
0.36d0          ;particle velocity inertia constant, w
0.01d0          ;chaos conditioning term, sigma  ... should not have to change
4863            ;random generator seed
600             ;number of iterations (generations), itrn
11              ;actual number of independent variables, m
3               ;ngoals
1.,1.,1.        ; xgls(j)
'kfuel 1'       ,  8.0d0       , 1.0d0        , 1.0d0        ;xmax,xmin,vlim
'rpvar 2'       ,  0.95d0      , 0.1d0        , 1.0d0        ;xmax,xmin,vlim
'rivar 3'       ,  0.99d0      , 0.01d0       , 1.0d0        ;xmax,xmin,vlim
'nsp 4'         ,  17.0d0      , 3.0d0        , 1.0d0        ;xmax,xmin,vlim
'fvar 5'        ,  0.2d0       , 0.01d0       , 1.0d0        ;xmax,xmin,vlim
'eps 6'         ,  0.9d0       , 0.1d0        , 1.0d0        ;xmax,xmin,vlim
'ptang 7'       ,  10.d0       , 1.0d0        , 1.0d0        ;xmax,xmin,vlim
'gl 8'          ,  800.d0      , 100.0d0      , 1.0d0        ;xmax,xmin,vlim
'rbi 9'         ,  24.d0       , 2.0d0        , 1.0d0        ;xmax,xmin,vlim
'diath 10'      ,  20.d0       , 0.5d0        , 1.0d0        ;xmax,xmin,vlim
'ratio 11'      ,  10.0d0      , 1.5d0        , 1.0d0        ;xmax,xmin,vlim
```

APPENDIX D: RPSO Source File

```
      subroutine swarm
c
c     SUBROUTINE TO FIND GLOBAL MINIMUM BY REPULSIVE PARTICLE SWARM METHOD
c     WRITTEN BY SK MISHRA, DEPT. OF ECONOMICS, NEHU, SHILLONG (INDIA)
c
c     modified by R.M. Jenkins, Auburn University (May-November 2009) to
c     include constrained optimization based on user-input parameter limits
c     and more realistic parameter behavior in multi-disciplinary
c     engineering optimization problems
c
      implicit double precision(a-h,o-z)
      parameter(np=100,nnp=25,mxp=100,nstepp=25,itrnp=2200,nsigmap=1)
      parameter(itopp=3,nprnt=100)
      parameter(npr=200,mstr=750,mpop=400,ngls=20)
      character names*14
      character ext*5,fname*80
      COMMON /RNDM/IU,IV
      common/swarmcontrol/minmax,ibest,mvar,iter,member,intl,intlans,
     &bstwrt,shtans
      common/bounds/xxmax(mxp),xxmin(mxp),vlim(mxp)
C      common/pass/yy(15,100)
      common/range/rangej(mxp)
      common/pop/amn(np)
      common/cal/ical
      common/best/bestans,bestyy(mxp)
      common/gls/ngoals
      common/gls2/xgls(ngls*2)
      INTEGER IU,IV
      dimension x(np,mxp),v(np,mxp),a(mxp),vi(mxp),tit(50),c(mxp)
      dimension xx(np,mxp),f(np),v1(mxp),v2(mxp),v3(mxp),v4(mxp)
      dimension bst(mxp),xmax(mxp),xmin(mxp),names(mxp+1)
      dimension cenj(mxp),xval(np,mxp),vval(np,mxp),z(np,mxp)
      dimension vlimin(mxp),vlimax(mxp),gbst(np,mxp),bestpop(itrnp)
      dimension trial(mxp)
      dimension xoa(itrnp,np,mxp),sol(np),bestx(itrnp,mxp)
      dimension fminmem(np),xbest(np,mxp)
      dimension bwt(mxp)
      data fmin /1.0d10/
      common/fcount/fcount
      integer fcount
      common/pass/yy(1,15,30),ys(11,40)
      integer pnlt(np),bstwrt
c
c     minmax          = 1 maximize; = 0 minimize
c     n               = population size
c     nn      = neighboring population sample size (must be less than n)
c     mx      = maximum allowable number of independent variables (f(x1,x2,......,mx)
c     nstep           = local tunneling parameter (ignores local gradients)
c              5 < nstep < 15
```

```
c     nprn  = results displayed every nprn iteration
c     nsigma is a perturbation parameter
c               nsigma = 0  no chaotic perturbation
c               nsigma = 1  chaotic perturbation
c     itop   = neighbor topology type
c               itop = 1    ring
c               itop = 2    ring and random
c               itop = 3    random
c     iu     = random number generator seed
c     itrn   = number of iterations (generations)
c     m                 = actual number of independent variables
c     xmax,xmin   = maximum and minimum parameter values; (xmax-xmin) = range
c     vlim       = upper limit on particle velocity, expressed as a
c                         multiplier on parameter range (maximum = 1.0)
c
      epsilon=1.D-20 ! ACCURACY NEEDED FOR TERMINATON
c
      open(unit=5,file='rpsoin.dat')
      open(unit=4,file='funcount.dat')
      open(unit=36,file='initpop.dat')
      open(unit=48,file='Best_data.dat')
      read(5,*)minmax
      read(5,*)n
      read(5,*)nn
      read(5,*)mx
      read(5,*)nstep
      read(5,*)nprn
      read(5,*)nsigma
      read(5,*)itop
      read(5,*)a1
      read(5,*)a2
      read(5,*)a3
      read(5,*)w
      read(5,*)sigma
      read(5,*)iu
      read(5,*)itrn
      read(5,*)m
      read(5,*) ngoals
      read(5,*) (xgls(j),j=2,ngoals+1)
c
      xgls(1)=float(ngoals)
      ys(4,5)=dble(float(ngoals))

      do j=1,m
      read(5,*) names(j),xmax(j),xmin(j),vlim(j)
            if(vlim(j).gt.1.0d0) vlim(j)=1.0d0
            xxmax(j)=xmax(j)
            xxmin(j)=xmin(j)
            enddo
            close(5)
      names(m+1)='fln'
      names(m+2)='z0'
      names(m+3)='tol'
      names(m+4)='parmatch'
c
      FFMIN=1.D30
      fmin=1.0d10
      LCOUNT=0
      npop=n
      mvar=m
c
c     generate boundaries for solution space
c
      iter=1
      fcount = 0
      avg = 0
      bstwrt = 0
      do j=1,m
              rangej(j)=xmax(j)-xmin(j)
      enddo
c
c     Generate an initial random population of m-tuple parameters x(i,j)
c     for "n" population members and calculate the fitness function of each member.
c     Each parameter is constrained to lie within user specified ranges.
c
      intl = 0
      pnlt = 0
      shtans = 10e9
      write(*,*)'filling initial population with viable solutions'
      isol=0
c
      write(36,714)(names(i),i=1,m),
     & ('fitness')
  714 format(7x,120(a14,1x))
c
   50 do 450 i=1,100000
        do j=1,m
                rangej(j)=xmax(j)-xmin(j)
          call random(rand)
                trial(j)=rand*rangej(j)+xxmin(j)
                a(j)=trial(j)
                ys(9,j)=a(j)
              enddo
                ys(4,2)=dble(float(isol+1))
                ys(4,3)=dble(float(0))
        call objective_function(a,M,solution)
              if(solution.lt.1.0d10.and.isol.lt.n.and.i.le.100000)then
                isol=isol+1
                write(*,*)i,isol,sngl(solution)
                sol(isol)=solution
                fminmem(isol)=solution
          do k=1,m
                  x(isol,k)=a(k)
                enddo
                if(isol.eq.n)then
           write(*,*)'initial population filled'
                 goto 60
                endif
              endif

  450 continue
c
```

```
      write(*,*)'initial population could not be completely filled
     &with viable solutions'
      do i=isol+1,n
      fminmem(i)=1.0d10
        do k=1,m
                call random(rand)
                x(i,k)= rand*rangej(k)+xmin(k)
              enddo
      sol(i) = 1.0d10
      enddo
   60 continue
c
c  write initial population data
      do i=1,n
      write(36,716)i,(x(i,ii),ii=1,m),
     & (sol(i))
  716 format(1x,i3,1x,120(e14.8,1x))
      enddo
      close(unit=36)
c  determine best member of initial population
      fmin=sol(1)
      bestans=sol(1)
      do j=1,m
       bestyy(j)=x(1,j)
      enddo
      ibest=1
      do i=2,n
              if(sol(i).lt.fmin)then
                fmin=sol(i)
                bestans=sol(i)
                do j=1,m
           bestyy(j)=x(i,j)
                enddo
                ibest=i
              endif
      enddo

      do j=1,n
              avg = avg + sol(j)
      enddo
      avg = avg/n

      best0=fmin                   !best value returned from random population
c
c     prepare data for next generation
c
      do i=1,n
              member=i
        amn(i)=fminmem(i)
        do j=1,m
                a(j)=x(i,j)
          z(i,j)=a(j)
c
c     randomize initial generation velocities
c
          call velocity(i,j,x,a1,dvelmin,dvelmax,velmin,velmax,rangevel)
        call random(rand1)
        vtemp=rand1*rangevel+velmin
              v(i,j)=vtemp-x(i,j)
            enddo
      enddo
c
      DO 100 ITER=1,ITRN                          !*** THIS IS THE GENERATION LOGIC LOOP ***
c
      ys(4,3)=dble(float(iter))
      if(fcount.gt.100000) goto 999
c
      write(ext,'(i5)') iter
      do kb=1,5
       if(ext(kb:kb).eq.' ')ext(kb:kb)='0'
      enddo
      fname='Iteration.'//ext
      open(unit=45,file=fname,status='unknown')
      write(45,714)(names(i),i=1,m),
     & ('fitness')
c
        do i=1,n                     !*** THIS IS THE POPULATION LOGIC LOOP ***
           member=i
           do j=1,m
            if(iter.eq.1)then
                 a(j)=z(i,j)
                else
                 a(j)=z(i,j)+v(i,j)            !array of all independent variables
c                                                           for particle "i" in a given generation
                endif
             c(j)=a(j)
             vi(j)=v(i,j)                  !array of velocities for particle "i"
           enddo
c
c       LSRCH lets each particle look in a limited "volume" of space
c       to determine if a better solution s available.  The array of parameters
c       for such a solution, if one exists, is called xx(member,i)
c
         amnn=amn(member)
         intl=1
c
              call lsrch(a,m,nstep,amnn,bst,fi)
   70      if(fi.lt.amnn)then
                 amn(member)=fi
               endif
           do in=1,m
            xx(member,in)=bst(in)
           enddo
         f(i)=fi
              write(45,716)i,(a(ii),ii=1,m),(f(i))
        enddo                                 !*** END OF POPULATION LOGIC LOOP ***
c
      if(iter.eq.1)then
         bestpop(iter)=f(1)
              ibest=1
         do i=2,n
                if(f(i).lt.bestpop(iter))then
```

```
                 bestpop(iter)=f(i)
                 ibest=i
               endif
            enddo
            fmin=bestpop(iter)
            bestoa=fmin
               member=1
               intl=2
               call setup
      endif
c
c     now consider all other iterations
c
      bestpop(iter)=f(1)
      ibest=1
      do i=2,n
            if(f(i).lt.bestpop(iter))then
              bestpop(iter)=f(i)
              ibest=i
            endif
      enddo
      if(bestpop(iter).lt.bestoa)then
        bestoa=bestpop(iter)
               member=1
               intl=2
               call setup
      endif
            fmin=bestpop(iter)

      do j=1,n
            avg = avg + f(j)
      enddo
      avg = avg/n

      write(27,*)'iteration no.',iter,'     fitness =',sngl(bestans)
      write(48,*)'iteration no.',iter,'     fitness =',sngl(bestans)
      write(*,*)'iteration no.',iter,'     fitness =',sngl(bestans)
      write(4,*) iter,fcount,sngl(bestans),avg
      avg = 0

      bstwrt = 1
      do l=1,m
      bwt(l) = bestyy(l)
      enddo
      call objective_function(bwt,M,solution)

      do j=1,m
      write(27,*)bestyy(j)
      write(48,*)bestyy(j)
      enddo
      bstwrt = 0
c
c     F(I) CONTAINS THE LOCAL BEST VALUE OF FUNCTION FOR ITH INDIVIDUAL
c
c     XX(I,J) IS THE M-TUPLE VALUE OF X ASSOCIATED WITH LOCAL BEST F(I)
c
```

```
c     NOW LET EVERY INDIVIDUAL RANDOMLY CONSULT NN(<<N) COLLEAGUES AND
c     FIND THE BEST AMONG THEM
c
     do i=1,n
c    ------------------------------------------------------------------
      IF(ITOP.GE.3) THEN
c    RANDOM TOPOLOGY ******************************************
c    CHOOSE NN COLLEAGUES RANDOMLY AND FIND THE BEST AMONG THEM
        BEST=1.0D10
         DO II=1,NN
              CALL RANDOM(RAND)
            NF=INT(RAND*N)+1
             IF(BEST.GT.F(NF)) THEN
              BEST=F(NF)
             NFBEST=NF
              ENDIF
         ENDDO
      ENDIF
C-------------------------------------------------------------------
      IF(ITOP.EQ.2) THEN
C     RING + RANDOM TOPOLOGY ******************************************
      BEST=1.0D10
        CALL NEIGHBOR(I,N,I1,I3)
        DO II=1,NN
             IF(II.EQ.1) NF=I1
              IF(II.EQ.2) NF=I
               IF(II.EQ.3) NF=I3
                   IF(II.GT.3) THEN
                  CALL RANDOM(RAND)
                   NF=INT(RAND*N)+1
                  ENDIF
               IF(BEST.GT.F(NF)) THEN
               BEST=F(NF)
               NFBEST=NF
              ENDIF
           ENDDO
      ENDIF
C------------------------------------------------------------------
      IF(ITOP.LE.1) THEN
C     RING TOPOLOGY **************************************************
      BEST=1.0D10
         CALL NEIGHBOR(I,N,I1,I3)
         do ii=1,3
             if(ii.eq.1) nf=i1
             if(ii.eq.2) nf=i
             if(ii.eq.3) nf=i3
             if(best.gt.f(nf)) then
               best=f(nf)
               nfbest=nf
             endif
           enddo
      endif
c-----------------------------------------------------------------
c    Each particle "i" will now move through the solution space (with a
c    "velocity" V) based on the following criteria
c
```

```
         do j=1,m
            vlimin(j)=-(x(i,j)-xxmin(j))
                  vlimax(j)=xxmax(j)-x(i,j)
               enddo
c
c     (1) velocity based on the particle's experience, including memory of
c         it's best position in the past ... xx(i)
c
      do j=1,m
        CALL RANDOM(RAND)
        V1(J)=A1*RAND*(XX(I,J)-X(I,J))
c
c     (2) velocity based on neighboring particles best experience, based on
c         the choice of solution topology ... "W" is an "inertial weight
c         parameter in the next three terms
c
        CALL RANDOM(RAND)
        V2(J)=V(I,J)
        IF(F(NFBEST).LT.F(I)) THEN
          V2(J)=A2*W*RAND*(XX(NFBEST,J)-X(I,J))
        ENDIF
c
c     (3) velocity selected randomly
c
        CALL RANDOM(RAND)
        RND1=RAND
        CALL RANDOM(RAND)
        V3(J)=A3*RAND*W*RND1
c
c     (4) velocity based on the particle's most recent velocity value
c
        V4(J)=W*V(I,J)
c
c     The total particle velocity is then
c
           V(I,J)= V1(J)+V2(J)+V3(J)+V4(J)
c
c      now ensure that the particle remains within the defined parameter space
c
               if(v(i,j).gt.0.0d0)then
                vmax=vlim(j)*vlimax(j)
                if(v(i,j).lt.vmax)then
                 goto 900
                else
                 v(i,j)=rand*vmax
          endif
               elseif(v(i,j).lt.0.0d0)then
                vmin=vlim(j)*vlimin(j)
                if(abs(v(i,j)).lt.abs(vmin))then
                 goto 900
                else
                 v(i,j)=rand*vmin
          endif
               endif
  900 continue
        X(I,J)=X(I,J)+V(I,J)
```

```
          ENDDO
       ENDDO
c
        DO I=1,n
          IF(F(I).LT.FMIN) THEN
            FMIN=F(I)
            II=I
            DO J=1,M
              BST(J)=XX(II,J)
            ENDDO
          ENDIF
        ENDDO
       IF(LCOUNT.EQ.NPRN) THEN
       LCOUNT=0
c       IF(DABS(FFMIN-FMIN).LT.EPSILON) GOTO 999
       FFMIN=FMIN
       ENDIF
       LCOUNT=LCOUNT+1
       ansitrn=fmin
              if(minmax.eq.1)ansitrn=1.0d0/fmin
  100 CONTINUE                                                   !***** END OF GENERATION LOGIC
LOOP *****
  999 continue
       ans=fmin
              if(minmax.eq.1)ans=1.0d0/fmin
              close(unit=4)
              return
  200 format(10x,100(a14,3x))
  201 format(4x,100(f10.4,8x))
  202 format(10x,102(a14,1x))
  203 format(/)
       end
c
       SUBROUTINE lsrch(a,m,nstep,amnn,bst,fi)
       implicit double precision(a-h,o-z)
       parameter(np=100,nnp=25,mxp=100,nstepp=25,itrnp=2200,nsigmap=1)
       COMMON /RNDM/IU,IV
       common/bounds/xxmax(mxp),xxmin(mxp),vlim(mxp)
c      common/pass/yy(15,100)
       common/range/rangej(mxp)
       common/swarmcontrol/minmax,ibest,mvar,iter,member,intl
       common/pop/amn(np)
       common/best/bestans,bestyy(mxp)
       common/pass/yy(1,15,30),ys(11,40)
       INTEGER IU,IV
       dimension a(mxp),b(mxp),x(np,mxp),xx(np,mxp)
       dimension c(500,mxp),bst(mxp)

c
c     This subroutine allows an individual particle to "wander" within a specified locality
c         to determine if a better solution can be found in that locality. It does so by
c         systematically varying each independent parameter sequentially.
c
              ichange=0
              amin=dfloat(-nstep/2)
              amax=dfloat(nstep/2-1)
```

```
c
c     "A" is the initial array of values for each independent variable for a given particle "i"
c     "B" is a new (possible) array of these variables, determined by NSTEP
c     "M" is the number of independent variables
c     "NSTEP" determines the separation of values examined within the local search volume,
c              centered on "A"
c     this subroutine is called for every particle "i" in a given generation
c
      ncount=1
      kbest=1
      fbest=amnn

              do k=1,m
                          b(k)=a(k)
              enddo

      do j=1,nstep                                        !step loop
c
         do jj=1,m                                        !loop systematically changes the parameter
c                                set sequentially

                          rangej(jj)=xxmax(jj)-xxmin(jj)
                          if(rangej(jj).lt.1d-5) then
                          b(jj)=xxmax(jj)
                          goto 350
                          endif

       call velocity(member,jj,x,vlim(jj),dvelmin,dvelmax,velmin,velmax,
     &   rangevel)
         anstep=dfloat(j-(nstep/2)-1)
         delamin=(dvelmin-0.0001d0)/(-amin)
               delamax=(dvelmax-0.0001d0)/amax
         if(anstep.lt.0.0d0)then

                call random(rand)

           b(jj)=a(jj)+delamin*anstep*rand
                 ys(9,jj)=b(jj)
                 call objective_function(b,m,fii)
           do k=1,m
             c(ncount,k)=ys(9,k)
                 enddo
                 goto 350
                 elseif(anstep.gt.0.0d0)then

              call random(rand)

           b(jj)=a(jj)+delamax*anstep*rand
                 ys(9,jj)=b(jj)
           call objective_function(b,m,fii)
           do k=1,m
             c(ncount,k)=ys(9,k)
                 enddo
                 goto 350
                 else
           ys(9,jj)=a(jj)
```

```
                 c(ncount,k)=ys(9,k)
                endif
  350 continue
        if(fii.lt.fbest)then
              fbest=fii                              !best value for member "i"from NSTEP
               if(fbest.lt.bestans)then
          bestans=fbest
                do jk=1,m
                 bestyy(jk)=c(ncount,jk)
                enddo
              endif
              amnn=fbest
              kbest=ncount
              endif
        amnn=fbest
        bst(jj)=c(kbest,jj)                                              !array of parameters associated with
c                                             best value of member "i" to date
              ncount=ncount+1
              enddo
     enddo
     fi=fbest
     return
     end
c
c    THIS SUBROUTINE IS NEEDED IF THE NEIGHBOURHOOD HAS RING TOPOLOGY
c    EITHER PURE OR HYBRIDIZED
c
      SUBROUTINE NEIGHBOR(I,N,J,K)
      IF(I-1.GE.1 .AND. I.LT.N) THEN
      J=I-1
      K=I+1
      ELSE
      IF(I-1.LT.1) THEN
      J=N-I+1
      K=I+1
      ENDIF
      IF(I.EQ.N) THEN
      J=I-1
      K=1
      ENDIF
      ENDIF
      RETURN
      END
c
     SUBROUTINE RANDOM(RAND1)
      DOUBLE PRECISION  RAND1
      COMMON /RNDM/IU,IV
      INTEGER IU,IV
      RAND=REAL(RAND1)
      IV=IU*65539
      IF(IV.LT.0) THEN
      IV=IV+2147483647+1
      ENDIF
      RAND=IV
      IU=IV
      RAND=RAND*0.4656613E-09
      RAND1=DBLE(RAND)
      RETURN
      END
c
     subroutine velocity(i,j,x,par,dvelmin,dvelmax,velmin,velmax,
    &rangevel)
c     subroutine to determine limits on particle velocity
     implicit double precision(a-h,o-z)
     parameter(np=100,nnp=25,mxp=100,nstepp=25,itrnp=2200,nsigmap=1)
     common/bounds/xxmax(mxp),xxmin(mxp),vlim(mxp)
     dimension x(np,mxp)
     dvelmin=dabs(x(i,j)-xxmin(j))
     dvelmax=dabs(xxmax(j)-x(i,j))
     velmin=x(i,j)-par*dvelmin
     velmax=x(i,j)+par*dvelmax
     rangevel=velmax-velmin
     return
     end
```

APPENDIX E: Compound RPSO Mini-Swarm Input File

```
5              ;population size, n2
5              ;neighboring population sample size, nn2 (must be less than n)
5              ;local sample space size, nstep2 (5 < nstep <15)
100            ;results displayed every "nprn" iteration, nprn2
1              ;chaos parameter, nsigma2 (=0 no chaotic perturbation, =1 chaotic perturbation)
3              ;neighborhood topology type, itop2 (=1 ring, =2 ring and random, =3 random)
0.25d0         ;particle velocity term 1 constant, a1_2
0.95d0         ;particle velocity term 2 constant, a2_2
0.05d0         ;particle velocity term 3 constant, a3_2
0.70d0         ;particle velocity inertia constant, w_2
0.05d0         ;percentage of solution space for miniswarm search, sp
20             ;number of iterations (generations), itrn2
```

APPENDIX F: Single Stage Solid Binary GA Input File

```
.false.                         ;micro  FIN
.false.                         ;pareto
.false.                         ;steady_state
.false.                         ;maximize
.true.                          ;elitist
.false.                          ;creep
.false.                          ;uniform
.false.                          ;restart
.true.                          ;remove_dup
.false.                          ;niche
.false.                          ;phenotype
0.04                            ;niche_diversity_percent_goal
67742                           ;iseed
0.9                             ;pcross
0.002                           ;pmutation
0.05                            ;pcreep
2                               ; ngoals
1.0 1.0                         ; xgls(j)
1.                              ;domst
2550                            ;convrg_chk(end_of_group2)
35                              ;no_para
 'rnos/rbod'    0.60000    0.40000    0.1000    .false. ;xmax_xmin_resolution_niche
 'lnos/dbod'    3.00000    1.50000    0.1000    .false. ;xmax_xmin_resolution_niche
 'kfuel___3'    9.00000    1.00000    1.0000    .false. ;xmax_xmin_resolution_niche
 'rpvar___4'    0.80000    0.30000    0.1000    .false. ;xmax_xmin_resolution_niche
 'rivar___5'    0.80000    0.10000    0.1000    .false. ;xmax_xmin_resolution_niche
 'nsp_____6'    11.0000    5.00000    1.0000    .false. ;xmax_xmin_resolution_niche
 'fvar____7'    0.10000    0.03000    0.0100    .false. ;xmax_xmin_resolution_niche
 'eps_____8'    0.95000    0.60000    0.0100    .false. ;xmax_xmin_resolution_niche
 'ptang___9'    10.00000   1.00000    1.0000    .false. ;xmax_xmin_resolution_niche
 'fn1____10'    0.99000    0.66000    0.0100    .false. ;xmax_xmin_resolution_niche
 'dth/Db_11'    0.30000    0.25000    0.0020    .false. ;xmax_xmin_resolution_niche
 'Lb/Db__12'    15.00000   10.0000    0.5000    .false. ;xmax_xmin_resolution_niche
 'dbody__13'    0.64000    0.25000    0.0020    .false. ;xmax_xmin_resolution_niche
 'b2w/DB_14'    0.05000    0.01000    0.0100    .false. ;xmax_xmin_resolution_niche
 'crw/DB_15'    0.05000    0.01000    0.0100    .false. ;xmax_xmin_resolution_niche
 'trw____16'    0.99000    0.90000    0.0100    .false. ;xmax_xmin_resolution_niche
 'wleswe_17'    30.0000    1.00000    1.0000    .false. ;xmax_xmin_resolution_niche
 'xLEw___18'    0.25000    0.20000    0.0100    .false. ;xmax_xmin_resolution_niche
 'b2t/DB_19'    1.40000    1.20000    0.1000    .false. ;xmax_xmin_resolution_niche
```

```
'crt/DB_20'   1.10000    0.90000    0.1000     .false. ;xmax_xmin_resolution_niche
'trt____21'   0.99000    0.50000    0.0100     .false. ;xmax_xmin_resolution_niche
'tleswp_22'   30.00000   1.00000    1.0000     .false. ;xmax_xmin_resolution_niche
'xTEt___23'   1.00000    0.95000    0.0100     .false. ;xmax_xmin_resolution_niche
'Apdly  24'   5000.0     4999.0     1.0        .false. ;xmax_xmin_resolution_niche
'thet0__25'   85.00000   40.00000   1.0000     .false. ;xmax_xmin_resolution_niche
'gainp1_26'   4.40000    3.60000    0.1000     .false. ;xmax_xmin_resolution_niche
'gainy1_27'   3.50000    1.00000    0.1000     .false. ;xmax_xmin_resolution_niche
'xcet___28'   .95000     .55000     0.0500     .false. ;xmax_xmin_resolution_niche
'dele0__29'   -7.000     -15.000    1.0000     .false. ;xmax_xmin_resolution_niche
'gainp2_30'   0.01000    0.00000    0.0100     .false. ;xmax_xmin_resolution_niche
'b2var__31'   0.01000    0.00000    0.0100     .false. ;xmax_xmin_resolution_niche
'dtchek_32'   1.00000    0.30000    0.1000     .false. ;xmax_xmin_resolution_niche
'gainy2_33'   0.01000    0.00000    0.0100     .false. ;xmax_xmin_resolution_niche
'delx-z_34'   00001.0    00000.0    00001.0    .false. ;xmax_xmin_resolution_niche
'delx-y_35'   00001.0    00000.0    00001.0    .false. ;xmax_xmin_resolution_niche
1                        ;frequency  FIN DEDR_6
200                      ;number of members in each generation
500                      ;number of generations
```

APPENDIX G: Single Stage Solid Real GA Input File

```
.false.                  ;generational  ONLY 1 GA TYPE CAN BE
.true.                   ;steady_state  ONLY 1 GA TYPE CAN BE
.false.                  ;hybrid        ONLY 1 GA TYPE CAN BE
.false.                  ;uniform x (50% parent1 and parent2
.true.                   ;Blend x (blend of parents)
.false.                  ;singlepointx
.false.                  ;var_mutation  true allows mutation
2000                     ;kcheck # of gen before 1/5 rule ck
0.2                      ;xmutation_rate  how much mutation
.1                       ;xmutation_amount % of variables mutated
2                        ;ngoals
1.0 1.0                  ;xgls(j)
35                       ;no_para
'rnos/rbod'     0.60000     0.40000     0.1000      .false. ;xmax_xmin_resolution_niche
 'lnos/dbod'    3.00000     1.50000     0.1000      .false. ;xmax_xmin_resolution_niche
 'kfuel___3'    9.00000     1.00000     1.0000      .false. ;xmax_xmin_resolution_niche
 'rpvar___4'    0.80000     0.30000     0.1000      .false. ;xmax_xmin_resolution_niche
 'rivar___5'    0.80000     0.10000     0.1000      .false. ;xmax_xmin_resolution_niche
 'nsp_____6'    11.0000     5.00000     1.0000      .false. ;xmax_xmin_resolution_niche
 'fvar____7'    0.10000     0.03000     0.0100      .false. ;xmax_xmin_resolution_niche
 'eps_____8'    0.95000     0.60000     0.0100      .false. ;xmax_xmin_resolution_niche
 'ptang___9'    10.00000    1.00000     1.0000      .false. ;xmax_xmin_resolution_niche
 'fn1____10'    0.99000     0.66000     0.0100      .false. ;xmax_xmin_resolution_niche
 'dth/Db_11'    0.30000     0.25000     0.0020      .false. ;xmax_xmin_resolution_niche
 'Lb/Db__12'    15.00000    10.0000     0.5000      .false. ;xmax_xmin_resolution_niche
 'dbody__13'    0.64000     0.25000     0.0020      .false. ;xmax_xmin_resolution_niche
 'b2w/DB_14'    0.05000     0.01000     0.0100      .false. ;xmax_xmin_resolution_niche
 'crw/DB_15'    0.05000     0.01000     0.0100      .false. ;xmax_xmin_resolution_niche
 'trw____16'    0.99000     0.90000     0.0100      .false. ;xmax_xmin_resolution_niche
 'wleswe_17'    30.0000     1.00000     1.0000      .false. ;xmax_xmin_resolution_niche
 'xLEw___18'    0.25000     0.20000     0.0100      .false. ;xmax_xmin_resolution_niche
 'b2t/DB_19'    1.40000     1.20000     0.1000      .false. ;xmax_xmin_resolution_niche
 'crt/DB_20'    1.10000     0.90000     0.1000      .false. ;xmax_xmin_resolution_niche
 'trt____21'    0.99000     0.50000     0.0100      .false. ;xmax_xmin_resolution_niche
 'tleswp_22'    30.00000    1.00000     1.0000      .false. ;xmax_xmin_resolution_niche
 'xTEt___23'    1.00000     0.95000     0.0100      .false. ;xmax_xmin_resolution_niche
 'Apdly  24'    5000.0      4999.0      1.0         .false. ;xmax_xmin_resolution_niche
 'thet0__25'    85.00000    40.00000    1.0000      .false. ;xmax_xmin_resolution_niche
 'gainp1_26'    4.40000     3.60000     0.1000      .false. ;xmax_xmin_resolution_niche
 'gainy1_27'    3.50000     1.00000     0.1000      .false. ;xmax_xmin_resolution_niche
 'xcet___28'    .95000      .55000      0.0500      .false. ;xmax_xmin_resolution_niche
```

```
'dele0__29'   -7.000     -15.000    1.0000    .false. ;xmax_xmin_resolution_niche
'gainp2_30'   0.01000    0.00000    0.0100    .false. ;xmax_xmin_resolution_niche
'b2var__31'   0.01000    0.00000    0.0100    .false. ;xmax_xmin_resolution_niche
'dtchek_32'   1.00000    0.30000    0.1000    .false. ;xmax_xmin_resolution_niche
'gainy2_33'   0.01000    0.00000    0.0100    .false. ;xmax_xmin_resolution_niche
'delx-z_34'   00001.0    00000.0    00001.0   .false. ;xmax_xmin_resolution_niche
'delx-y_35'   00001.0    00000.0    00001.0   .false. ;xmax_xmin_resolution_niche
  1                      ; ifreq FIN DEDR_6
  30                     ; mempops
  100000                 ; maxgen
```

APPENDIX H: Single Stage Solid RPSO Input File

```
0             ;maximize objective function = 1, minimize objective function = 0
30            ;population size, n
10            ;neighboring population sample size, nn (must be less than n)
100           ;maximum allowable number of independent variables, mx
5             ;local sample space size, nstep (5 < nstep <15)
100           ;results displayed every "nprn" iteration, nprn
1             ;chaos parameter, nsigma (=0 no chaotic perturbation, =1 chaotic perturbation)
3             ;neighborhood topology type, itop (=1 ring, =2 ring and random, =3 random)
0.875d0       ;particle velocity term 1 constant, a1
0.75d0        ;particle velocity term 2 constant, a2
0.625d0       ;particle velocity term 3 constant, a3
0.36d0        ;particle velocity inertia constant, w
0.01d0        ;chaos conditioning term, sigma  ... should not have to change
4863          ;random generator seed
2200          ;max number of iterations (generations), itrn
35            ;actual number of independent variables, m
2             ; ngoals
1.0 1.0       ; xgls(j)
 'rnos/rbod'  0.60000      0.40000      1.0d0      ;xmax,xmin,vlim
 'lnos/dbod'  3.00000      1.50000      1.0d0      ;xmax,xmin,vlim
 'kfuel___3'  9.00000      1.00000      1.0d0      ;xmax,xmin,vlim
 'rpvar___4'  0.80000      0.30000      1.0d0      ;xmax,xmin,vlim
 'rivar___5'  0.80000      0.10000      1.0d0      ;xmax,xmin,vlim
 'nsp_____6'  11.0000      5.00000      1.0d0      ;xmax,xmin,vlim
 'fvar____7'  0.10000      0.03000      1.0d0      ;xmax,xmin,vlim
 'eps_____8'  0.95000      0.60000      1.0d0      ;xmax,xmin,vlim
 'ptang___9'  10.00000     1.00000      1.0d0      ;xmax,xmin,vlim
 'fn1____10'  0.99000      0.66000      1.0d0      ;xmax,xmin,vlim
 'dth/Db_11'  0.30000      0.25000      1.0d0      ;xmax,xmin,vlim
 'Lb/Db__12'  15.00000     10.0000      1.0d0      ;xmax,xmin,vlim
 'dbody__13'  0.64000      0.25000      1.0d0      ;xmax,xmin,vlim
 'b2w/DB_14'  0.05000      0.01000      1.0d0      ;xmax,xmin,vlim
 'crw/DB_15'  0.05000      0.01000      1.0d0      ;xmax,xmin,vlim
 'trw____16'  0.99000      0.90000      1.0d0      ;xmax,xmin,vlim
 'wleswe_17'  30.0000      1.00000      1.0d0      ;xmax,xmin,vlim
 'xLEw___18'  0.25000      0.20000      1.0d0      ;xmax,xmin,vlim
 'b2t/DB_19'  1.40000      1.20000      1.0d0      ;xmax,xmin,vlim
 'crt/DB_20'  1.10000      0.90000      1.0d0      ;xmax,xmin,vlim
 'trt____21'  0.99000      0.50000      1.0d0      ;xmax,xmin,vlim
 'tleswp_22'  30.00000     1.00000      1.0d0      ;xmax,xmin,vlim
 'xTEt___23'  1.00000      0.95000      1.0d0      ;xmax,xmin,vlim
```

```
'Apdly  24'    5000.0      4999.0      1.0d0       ;xmax,xmin,vlim
'thet0__25'    85.00000    40.00000    1.0d0       ;xmax,xmin,vlim
'gainp1_26'    4.40000     3.60000     1.0d0       ;xmax,xmin,vlim
'gainy1_27'    3.50000     1.00000     1.0d0       ;xmax,xmin,vlim
'xcet___28'    .95000      .55000      1.0d0       ;xmax,xmin,vlim
'dele0__29'    -7.000      -15.000     1.0d0       ;xmax,xmin,vlim
'gainp2_30'    0.01000     0.00000     1.0d0       ;xmax,xmin,vlim
'b2var__31'    0.01000     0.00000     1.0d0       ;xmax,xmin,vlim
'dtchek_32'    1.00000     0.30000     1.0d0       ;xmax,xmin,vlim
'gainy2_33'    0.01000     0.00000     1.0d0       ;xmax,xmin,vlim
'delx-z_34'    00001.0     00000.0     1.0d0       ;xmax,xmin,vlim
'delx-y_35'    00001.0     00000.0     1.0d0       ;xmax,xmin,vlim
```