Applied Mathematics and Computation 218 (2012) 4710–4730

Contents lists available at ScienceDirect

Applied Mathematics and Computation

journal homepage: www.elsevier.com/locate/amc

Aerospace design optimization using a steady state real-coded genetic algorithm

John D. Dyer [a,1], Roy J. Hartfield [a,*,2], Gerry V. Dozier [b,3], John E. Burkhalter [a,4]

[a] Auburn University, Auburn, AL 36849, United States
[b] North Carolina A&T, Greensboro, NC 27411, United States

ARTICLE INFO

Keywords:
Aerospace
Optimization
Rocket propulsion
Genetic algorithm
Real coded genetic algorithm
Steady state genetic algorithm

ABSTRACT

This study demonstrates the advantages of using a real coded genetic algorithm (GA) for aerospace engineering design applications. The GA developed for this study runs steady state, meaning that after every function evaluation the worst performer is determined and that worst performer is then thrown out and replaced by a new member that has been evaluated. The new member is produced by mating two successful parents through a crossover routine, and then mutating that new member. For this study three different preliminary design studies were conducted using both a binary and a real coded GA including a single stage solid propellant missile systems design, a two stage solid propellant missile systems design and a single stage liquid propellant missile systems design.

1. Introduction

Optimization of aerospace engineering applications using genetic algorithms (GA's) such as spacecraft controls [1,2], turbines [3], helicopter controls [4], flight trajectories [5], wings and airfoils [6,7], missiles [8–11], rockets [12], propellers [13] and inlets [14] have been preformed with great success. In some cases, real coded GA's have been shown to produce better results than binary coded GA's [15–17]. This success is the primary motivation for the current study involving aerospace applications. The goal of engineering design optimization is to find an optimum solution to a design problem. Optimization has evolved throughout the years from classical methods to modern evolutionary algorithms. Modern computers with their exceedingly fast computational times have enabled optimizers to become extremely efficient at solving very complex problems.

All GA's are based on the principles developed by John Holland in his book ‘‘adaptation in natural and artificial systems'' [18]. Holland outlined the methods for successfully implementing population based adaptive optimizers. Holland's methods operate on the principle of survival of the fittest. In a computational sense, candidate solutions are assembled in a population and compared to one another, the weak die off and the strong are left to reproduce and mutate to produce better children.

The search for a more efficient optimizer for some of these aerospace applications arose due to the extended run times associated with long range missiles using the IMPROVE© optimizer (Implicit Multi-objective PaRameter Optimization Via Evolution) [19]. The IMPROVE© binary encoded generational GA has been used extensively for optimization of missile systems and has been shown to be a very versatile and robust method for optimization. A primary disadvantage of the binary

* Corresponding author.
E-mail address: rjh@eng.auburn.edu (R.J. Hartfield).
[1] Graduate Research Assistant, Aerospace Engineering, AIAA Member.
[2] Professor, Aerospace Engineering, Associate Fellow AIAA.
[3] Department Chair, Computer Science.
[4] Professor Emeritus, Aerospace Engineering, Associate Fellow AIAA.

0096-3003/$ - see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.amc.2011.07.038

Nomenclature

μ	mutation rate
σ	mutation amount
Ae	nozzle exit area
A^*	nozzle throat area
b2tail	tail semi-span
b2wing	wing semi-span
crtail	tail root chord
crwing	wing root chord
dbody	diameter of body
dnose	nose diameter
dstar	throat diameter
eps	epsilon-grain
f	propellant grain fillet radius
GA	genetic algorithm
lbody	length of body
LE	leading edge
lnose	length of nose
N	number of star points-grain
rbody	radius of body
Ri	propellant inner grain radius
rnose	radius of nose
Rp	propellant outer grain radius
TE	trailing edge
TOF	time of flight
xLEt	X-location of tail leading edge
xLEw	X-location of wing leading edge

coded GA comes from the fact that because all of the variables must be converted into a single bit string, the solution accuracy is dependant on the number of bits that can be used for the string.

Because of the large ranges associated with many of the design parameters the smallest resolution for the binary GA is generally limited to about 1% of the solution space for complex problems while the real coded GA is only limited to a double precision number. The two stage solid missile system model used in this study has 46 design parameters making up each member of the population, thereby compounding the resolution problems for the binary GA. Resolution is not a significant issue with a real coded GA because all of the variables remain real double precision variables. Dozier et al. [20], Unsal et al. [21] and Dozier et al. [22] demonstrated the ability for a real coded GA to achieve shorter run times as well as more accurate solutions for some applications. Hamming cliffs can also pose a problem for the binary GA because all of the design parameters are converted into a single bit string. For example if two integers 15, and 16 were represented by the bit strings 01111 and 10000 respectively , the GA would have to change all of the bits simultaneously to change from 15 to 16. Mutation and crossover do not always solve this problem. Hamming cliffs are not possible with the real GA because the design variables remain real coded.

Another advantage of the real GA created for this study is that it uses steady state optimization unlike the generational optimization used by the binary GA. The key difference is that in the steady state GA for each generation only the worst performer is thrown out and replaced by a new member, whereas for the generational GA all of the members of the population are thrown out and replaced (expect in elitist mode when the best member remains in the next generation) using a similar tournament routine. For complex problems the steady state GA may not be as efficient as the generational GA because of its lack of diversity. For the steady state GA, once the survivors have had a chance to crossover (i.e. pass genetic material back and forth through their variables), the new member replacing the worst performer is run back through the objective function. This process continues, with the parents producing on average better offspring, until the maximum number of generations (user specified) is reached. There are proofs [23,24] which show why this process produces increasingly superior performers in a population, but a simplistic view is that a good parent mated with another good parent, is more likely to produce good offspring than two poor parents when mated. This is not to say that two good parents cannot produce poor performers. Rather, when two good performers exchange genes, statistically the resulting offspring have a higher chance of outperforming their parents.

2. Real-coded GA methodology

The real and binary GA's used in this study both operated using the same tournament style evolution of a population. They each work with a number of candidate solutions to solve a particular problem. A data structure known as an individual is used to represent each candidate solution. Each individual has a fitness and a chromosome. Each chromosome is made up of genes, in this case the genes used are the GA variables in the design variable input (GANNL.DAT) files associated with each code used. A group of individuals makes up a population. In order to create a new population, individuals called parents are selected based on their fitness and allowed to create children using a tournament selection process. Details of the binary GA operation can be found in reference 19. For the real coded GA, parents are chosen in groups of two and the parent with the better fitness survives to produce children. This process is repeated to select a second parent. The tournament selection process is shown in Fig. 1. Both parents then use a crossover routine to mix their genes in order to produce a child.

2.1. Crossover

Three different crossover routines were used for the real coded GA, Blend X, uniform and singlepoint crossover. For Blend X crossover [25] each design parameter of the two parents selected is subtracted and then multiplied by a random number (between 0 and 1), and added to the smaller parent as shown in Eq. (1) and Fig. 2. Blend X crossover allows for the most mutation of all the crossover routines. Blend X crossover creates children unique from their parents, unlike uniform or single-point crossover, whose children are the same values from either one parent or the other. It should be noted while Blend X crossover does produce a unique child from the two parents, the child cannot be outside the range of the values of the parents, and for this reason mutation is necessary.

For $i = 1$ to number of design parameters:

$$\text{Child}(i) = abs(\text{Parent1}(i) - \text{Parent2}(i)) \times \text{Random\#} + \min(\text{Parent1}(i) - \text{Parent2}(i)). \tag{1}$$

Both single-point and uniform crossover swap design parameters between parents. For single-point crossover a random number is used to define a cut point where the first parents design parameters are used up to that cut point, and the second parents design parameters are used after that cut point, as shown in Fig. 3.

Uniform crossover works very similarly to single-point crossover except each design parameter in the child is randomly chosen from either parent 1 or parent 2 this allows a child to be made up of any combination of both parents design parameters as shown in Fig. 4.

2.2. Mutation

After a child has been created it can be mutated using a Gaussian mutation routine. Two main operators control the mutation, mutation rate and mutation amount. Mutation rate, μ determines how many genes of each child will be mutated, this operator is set between 0.0 and 1.0, with 1.0 mutating every gene within the child. The second operator used in mutation is

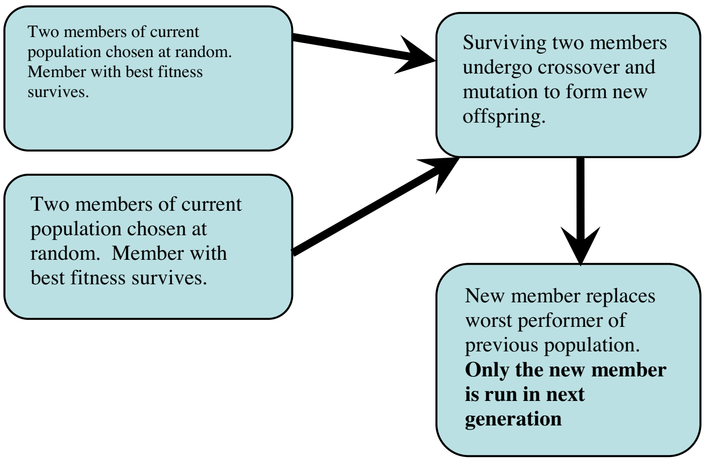

Two members of current population chosen at random. Member with best fitness survives.
Two members of current population chosen at random. Member with best fitness survives.
Surviving two members undergo crossover and mutation to form new offspring.
New member replaces worst performer of previous population. **Only the new member is run in next generation**

Fig. 1. Tournament selection mutation and crossover flow chart.

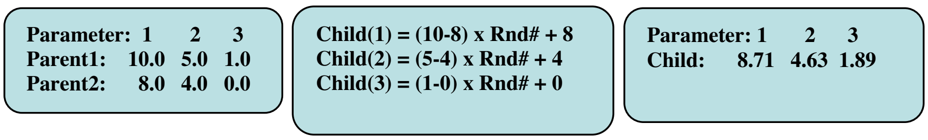
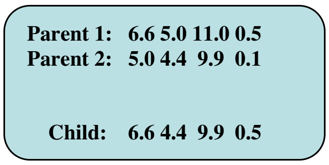
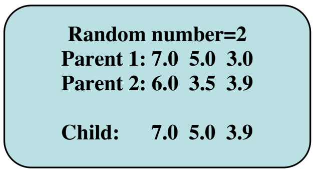


Fig. 2. Blend X crossover.



Parent 1:	6.6 5.0 11.0 0.5
Parent 2:	5.0 4.4 9.9 0.1
Child:	6.6 4.4 9.9 0.5

Fig. 3. Single-point crossover.



Random number=2

Parent 1:	7.0 5.0 3.0
Parent 2:	6.0 3.5 3.9
Child:	7.0 5.0 3.9

Fig. 4. Uniform crossover.

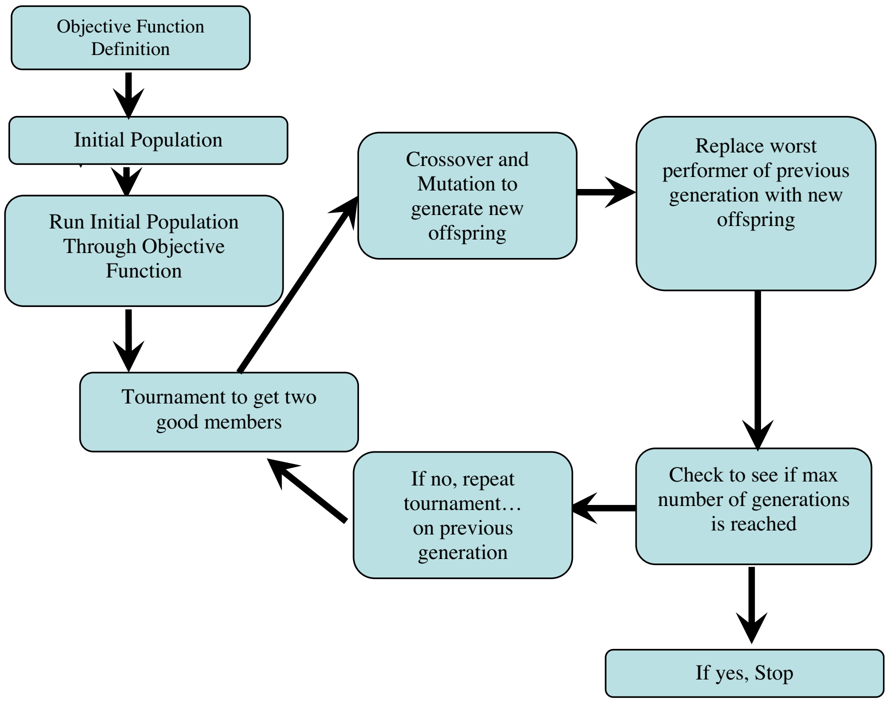

Objective Function Definition
Initial Population
Run Initial Population Through Objective Function
Tournament to get two good members
Crossover and Mutation to generate new offspring
Replace worst performer of previous generation with new offspring
Check to see if max number of generations is reached
If no, repeat tournament… on previous generation
If yes, Stop

Fig. 5. Real coded steady state GA program flow.

the mutation amount σ. The mutation amount determines how much each gene will be mutated, this usually ranges from 0.5 (high mutation) to 0.005 (very low mutation). A Gaussian distributed random number is used with the mutation amount in order to mutate the child as shown in Eq. (2).

For $i = 1$ to number of genes:

$$\text{Child}(i) = \sigma^{*}[x\max(i) - x\min(i)]^{*} \textit{gaussian\_random\_number} + \text{Child}(i). \tag{2}$$

Where xmax (i) and xmin (i) are the maximum and minimum values specified by the user for each gene in the GA variable input file (GANNL.DAT). If any of the mutated child's design parameters are larger than the maximum or smaller than the minimum value then that design parameter is set to the maximum or the minimum value. It has been shown that mutating with a Gaussian distributed random number with a zero mean and unit variance works well for optimizations [26].

After the parents design parameters have been crossed and mutated to produce a new child or individual, that individual is evaluated and given a fitness value. In order for the new individual to become a member of the population, one member must die in order for the new member to take its place. The member that dies off is the member in the population with the worst fitness. This method of allowing individuals to create new children while also killing off members with bad fitness' is known as natural selection [27]. This process of creating new children and killing off members with bad fitness' continues on until a set number of iterations have been satisfied. The entire program flow for the real coded GA is shown in Fig. 5.

2.3. Steady state GA

The real GA developed for this study operates in steady state as opposed to generational as the binary GA. The main difference between steady state and generational is the number of members after each generation that are evaluated by the objective function. For a generational GA all of the members of a population, are evaluated by the objective function. Their fitness values are then evaluated and sorted and then a tournament scheme replaces each member of the population by crossover and mutation, and all of the new members are evaluated by the objective function, except for perhaps the best performer if elitism has been employed.

For a steady state GA after the initial population is evaluated by the objective function, only one member is replaced by a tournament scheme using crossover and mutation for each generation. That new member is evaluated by the objective function and then replaces the previous generation's worst performer. This can lead to quick and very accurate convergence due to the fact that the member immediately becomes part of the mating pool making a shift toward an optimal fitness possible early in the optimization [28]. A flow chart showing the steady state process in detail is shown in Fig. 6. For the generational GA the entire gene pool is replaced by children that have been created by mating successful parents thus giving the large

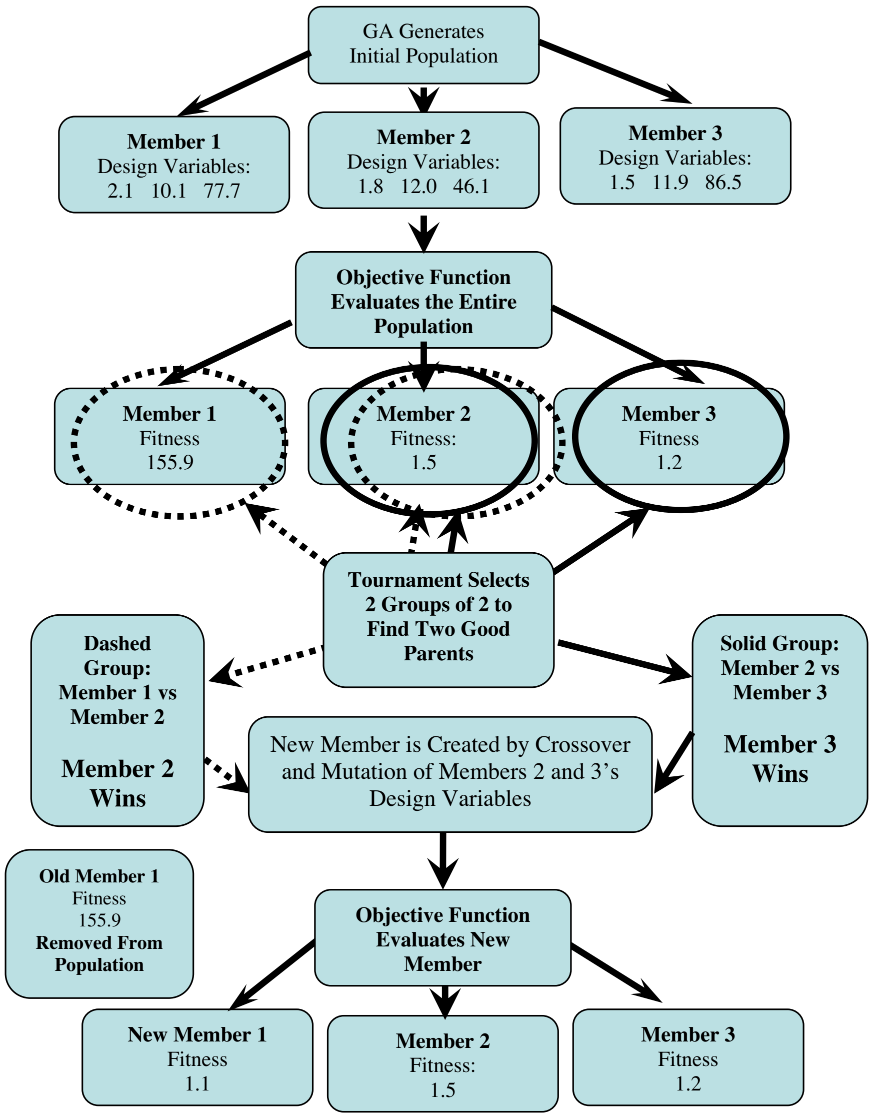

GA Generates Initial Population
Member 1 Design Variables: 2.1 10.1 77.7
Member 2 Design Variables: 1.8 12.0 46.1
Member 3 Design Variables: 1.5 11.9 86.5
Objective Function Evaluates the Entire Population
Member 1 Fitness 155.9
Member 2 Fitness: 1.5
Member 3 Fitness 1.2
Tournament Selects 2 Groups of 2 to Find Two Good Parents
Dashed Group: Member 1 vs Member 2
Member 2 Wins
Solid Group: Member 2 vs Member 3
Member 3 Wins
New Member is Created by Crossover and Mutation of Members 2 and 3's Design Variables
Old Member 1 Fitness 155.9 Removed From Population
Objective Function Evaluates New Member
New Member 1 Fitness 1.1
Member 2 Fitness: 1.5
Member 3 Fitness 1.2

Fig. 6. Steady state detailed flow chart.

amount of diversity associated with the generational GA. The main drawback to steady state is that it does not have the large number of random guesses that the generational GA can obtain. For the steady state GA only one member of the gene pool is replaced for each generation, and that member is the worst performer of the previous generation. Since the member that is created is composed of two good members from the previous generation the steady state GA can quickly converge to a good solution, however if none of the members of the initial population are good members the steady state GA can only rely on mutation of the one member each generation to find a good fitness. For this reason BlendX crossover was included to enable more mutation for each new member.

3. Missile system descriptions

Three different missile system design codes were used in this study to compare the real and binary GA's. The missile system design code creates and flies preliminary design level models for missiles powered by one or two stage solid propellant rocket propulsion or liquid rocket propulsion using a set of approximately 26–46 critical design variables. The GA is used to optimize the missile to meet certain goals. Burkhalter et al. have developed the programs used to design the physical model of the missile that they described in their paper "Missile Systems Design Optimization Using Genetic Algorithms" [10]. Further research has been conducted on missile design optimization by Hartfield et al. in their paper, "Optimizing a Solid Rocket Motor Boosted Ramjet Powered Missile Using a Genetic Algorithm" [11]. The missile design codes have been shown to be a very good tool for preliminary design and have been extensively used as a design tool.

All three missile system design codes follow a similar program flow. In the main program, the first task is to read in two input files that contain constants including densities, masses and moments of inertia. The block of information in these files is divided into major sections as listed in Table 1. The number listed to the right is the number of variables included in each of the major sections. Initially, the table must be generated with known values for each of the variables.

After the initial constants input files are read in, the GA input file which contains the GA variables to be optimized is read. For each member of a population a new missile is built using the data from the initial constants input file and the design parameters that the GA generates using the GA input file. After the initial parameters are stored, the program designs and flies the missile using a series of subroutines to determine the propulsion, mass properties and aerodynamics. This missile is then flown in the 6-DOF routine as shown in Fig. 7.

In order to fly the missile through the 6-dof the propulsion of the missile must be determined along with the masses and finally the missile aerodynamics. The thrust of the missile is determined using the propellant parameters. For both solid propellant systems, the part of the thrust equation which is not dependant on atmospheric pressure is determined from the grain geometry specified in the propellant column of Table 2. Tail-off for the thrust after burnout is modeled because of the large cavity volume at the end of the burn after the propellant has been ejected. For the liquid system the pressure and temperature inside the combustion chamber are assumed to be constant, also the thrust is assumed to go to zero as soon as all of the fuel has been consumed. Unlike the solid systems there is no computation of a thrust tail-off. After the thrust is determined the mass properties subroutine is used to determine the center of gravity and moments of inertia for each component of the missile systems.

The aerodynamics are the last parameters to be calculated before the missile can be flown through the 6-dof. In order to obtain reasonably short run times a fast predictive aerodynamics code was implemented, Aerodsn. Aerodsn is used in order to generate the aerodynamics for each missile code. While the code is non-linear, many approximations are made concerning the flow field of the missile. All flows are assumed to be attached with no boundary layers. After the missile has been designed and the aerodynamics have been calculated the missile is ready to be flown through the 6-dof. The 6-dof

Table 1
General form of the initial constants input file for the missile design codes.

1	Densities 30
2	Masses 30
3	Center of gravity 27
4	Moments of inertia 60
5	Lengths and limits 30
6	External geometry 30
7	Required computed data from aero 30
8	Other dimensions 11
9	Internal solid rocket grain variables 12
10	Nozzle and throat variables 14
11	Other computed stage variables 8
12	Program lengths,limits and constants 10
13	Initiation of launch data 14
14	Target data 6
15	GA goals (outdata variables) 20
16	Auxillary variables to be used as needed 10
17	List of GA variables passed to setup, etc. 29
18	Total missile variables 10

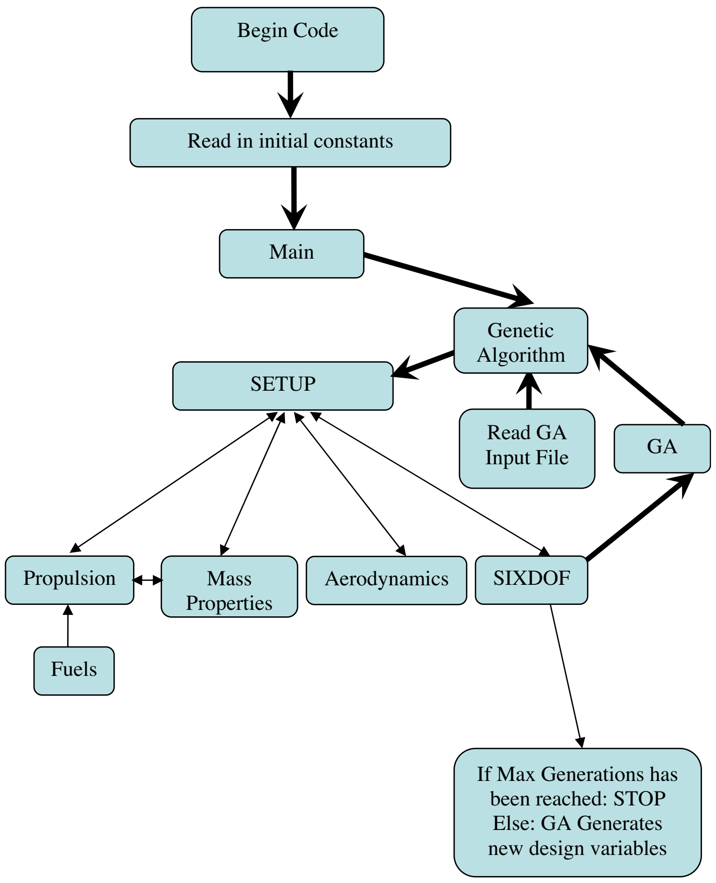

Begin Code
Read in initial constants
Main
Genetic Algorithm
SETUP
Read GA Input File
GA
Propulsion
Mass Properties
Aerodynamics
SIXDOF
Fuels
If Max Generations has been reached: STOP Else: GA Generates new design variables

Fig. 7. Missile system design code program flow.

Table 2
Single stage solid GA variable definition.

Geometry	Propellant	Auto pilot
Nose radius ratio = rnose/rbody	Fuel type	Auto pilot on delay time
Nose length ratio = Lnose/dbody	Propellant out rad rpvar = (rp + f)/rbody	Auto pilot time constant-tau, f.c.s. time const
Fractional nozzle length ratio = f/rp	Propellant inner radius ratio = ri/rp	Auto pilot damping – zeta, f.c.s. damping
Throat diameter/Dbody	Number of star points	Cross over freq – wcr, f.c.s. crossover freq
Total length of stage1/Dbody	Fillet radius ratio = f/rp	Pronav gain – guidance gain
Dia of stage1 center section	Epsilon – star width	
Wing exposed semi-span = b2w/dbody	Star point angle	
Wing root chord = crw/dbody		
Wing taper ratio = ctw/crw		
LE sweep angle deg		
xLEw/lbody		
Tail exposed semi-span = b2t/dbody		
Tail root chord = crt/dbody		
Tail taper ratio = ctt/crw		
LE sweep angle deg		
xTEt/lbody		
Initial launch angle (deg)		

implemented in the missile codes uses a spherical earth model with all six degrees of freedom modeled for a ballistic flight. After the missile has been flown through the 6-dof the performance is then stored and used to compute the overall fitness depending on the goal chosen for the optimization.

The first system tested was the single stage solid missile design. The single stage solid missile design uses 29 design variables in order to produce a physical model of the system. An overview of the 29 design variables is shown in Table 2. These variables are used by the GA in conjunction with the initial constants to generate a solid propellant grain as shown in Fig. 8.

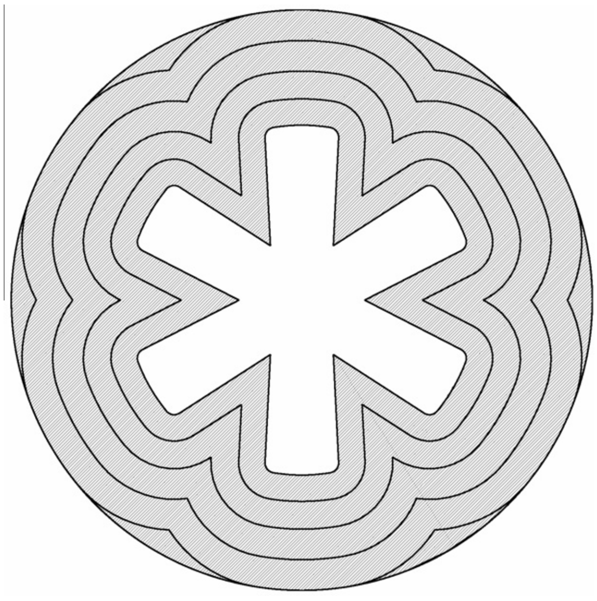


Fig. 8. Solid propellant grain cross-section.

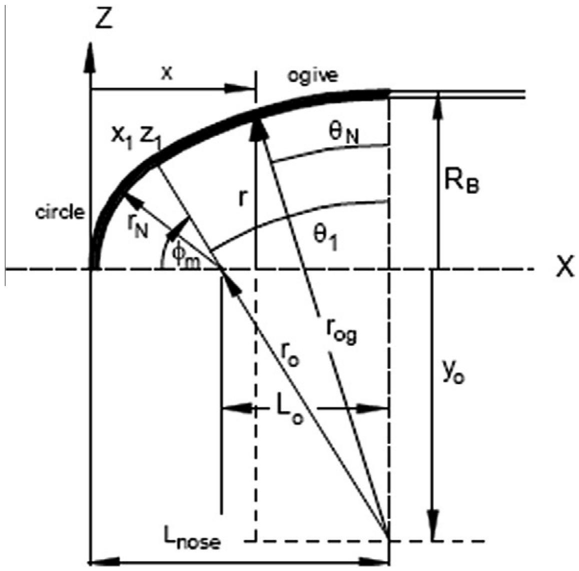


Fig. 9. Nose design schematic.

For the two solid systems circularly perforated grains, star grains and wagon wheel grains are considered. The nose, nozzle, tail and all remaining geometries are modeled using the geometry column of Table 2. The nose for each missile is generated using the geometry as shown in Fig. 9. The geometry also used to physically design the nozzle and tails as shown in Figs. 10 and 11. The nozzle designed for each missile code is a bell nozzle. To achieve the "bell" shape, a parabola is fitted tangent to a circular arc throat section and terminates at the exit plane with the required expansion ratio [29].

The second system tested was the two stage solid propellant missile design. The two stage solid missile design operates in a similar fashion to the single stage solid code, however it uses 46 design variables in order to produce a physical model of the system instead of the 29 used in the single stage solid code.

The third missile design code tested was the liquid propulsion missile design code. The liquid code follows a slightly different program flow than the two solid codes. The basic performance of the liquid code is based on the assumption that combustion takes place under constant pressure and constant temperature. The initial constants input file is set up similarly to the solid input files, however the GA variables input file is comprised of a different set of 26 design parameters. An overview of the 26 design variables is shown in Table 3. For the liquid system it is assumed to have a single stage comprised of a single engine, where the fuel and oxidizer tanks are modeled to be cylindrical tanks with hemispherical endcaps.

The geometry for the liquid system is used to generate the physical model of the missile in the same way that it was developed for the two solid systems as shown in Fig. 12. The main difference between the solid systems and the liquid

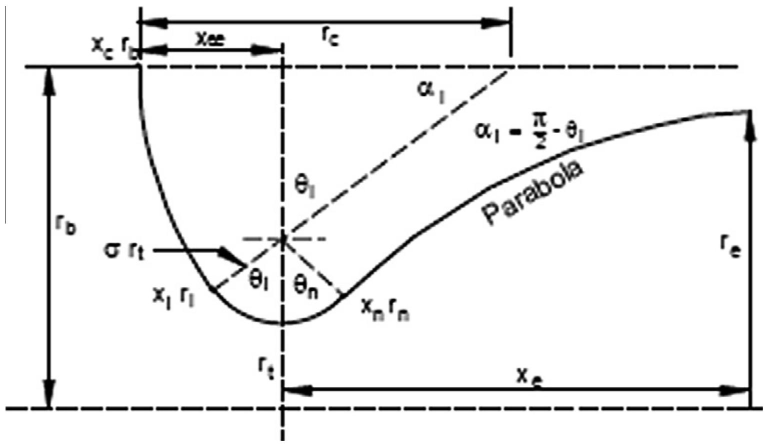


Fig. 10. Nozzle design schematic [29].

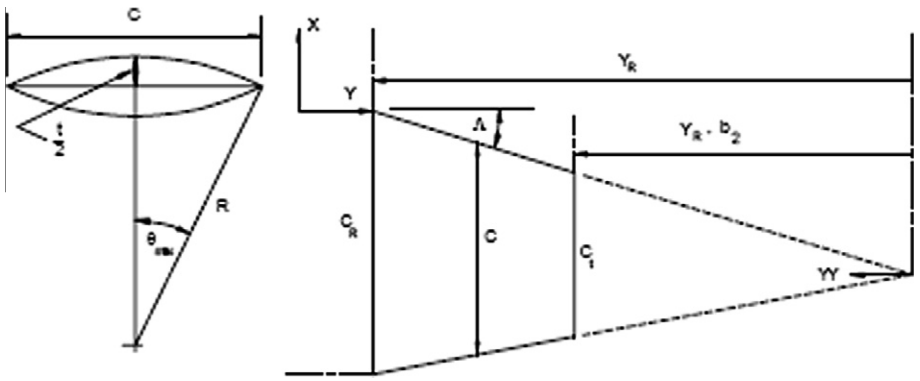


Fig. 11. Tail design schematic.

Table 3
Single stage liquid GA variable definition.

Geometry	Propellant	Auto pilot
Body diameter	Propellant type	Auto pilot on delay time
Nose dia ratio = dnose/DB	Oxidizer to fuel ratio	Auto pilot time constant-tau, f.c.s. time const
Length of nose ratio = blnose/dnose	Chamber pressure	Auto pilot damping – zeta, f.c.s. damping
Nozzle throat dia ratio = dstar/dbody		Cross over freq – wcr, f.c.s. crossover freq
Nozzle expansion ratio = Ae/A*		Pronav gain – *n*-prime, guidance gain
Fractional nozzle length		
Wing root chord ratio = crwing/DB		
Wing taper ratio		
Wing b/2 ratio = b2wing/DB		
Wing le angle		
Wing X loc = xLEwing/totlen		
Tail root chord ratio = crtail/DB		
Tail taper ratio		
Tail b/2 ratio = b2tail/DB		
Tail le angle deg		
Tail X loc = xTEtail/totlen		
Burn time		
Initial launch angle (deg)		

systems is the propellant. For the liquid system, the code generated models of the tanks used to store the fuel and oxidizer. The second column in Table 3 is used to determine the fuel-oxidizer combination as well as the initial chamber pressure. These inputs along with the geometric inputs are used to design the liquid propellant missile.

4. Real GA convergence testing

The binary GA IMPROVE© used in this study has been used for many years and has proven to be very robust. Because the binary GA has been proven to be successful with its current GA parameters, no testing was done in order to optimize the binary GA parameters, for this study they remained constant. In order to compare the binary and the real GA's, the parameters for the real GA needed to be tested. The parameters tested were the type of crossover (blendx, single-point and uniform), mutation rate, mutation amount and population size. Changing any one of these parameters can cause the GA to converge faster or slower to a solution. For these tests the single stage solid missile system design code was used to match a range of 400,000 ft and a weight of 2500 lbm.

The first parameter tested was crossover. Each type of crossover was tested using the single stage solid code and run out to 10,000 function evaluations to determine which type would achieve the best fitness. All three types of crossovers performed well in this test, with the BlendX converging more rapidly to the best fitness as seen in Fig. 13. This coupled with

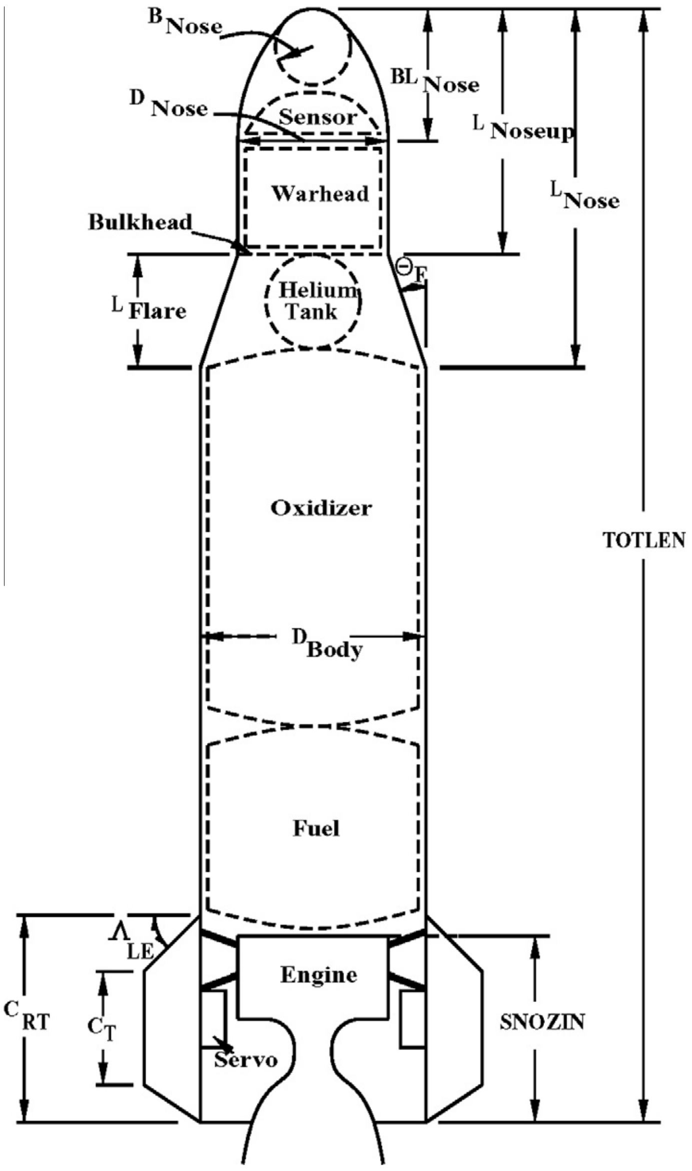


Fig. 12. Liquid missile design schematic.

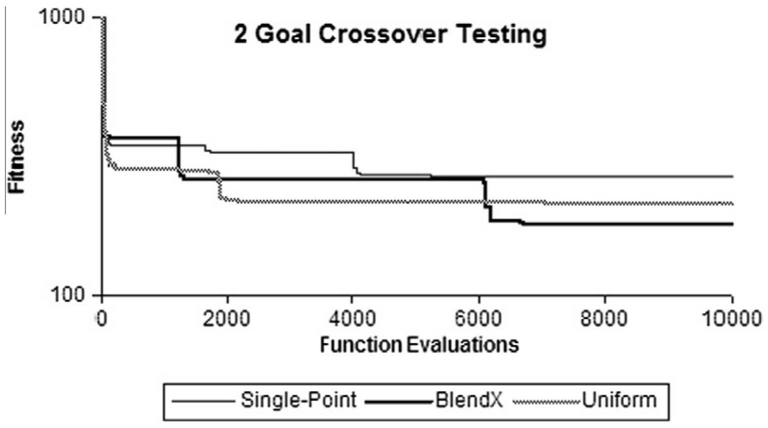


Fig. 13. One stage solid missile crossover testing 400,000 ft (two) goal.

Table 4
Mutation operator fitness comparison.

Two goal mutation testing		Mutation amount	Fitness
Mutation operators	Mutation rate		
Test_1	1.00	0.05	190.49
Test_2	0.20	0.10	181.33
Test_3	0.05	1.00	288.03
Test_4	0.50	0.05	190.25

the fact that the binary GA had single-point crossover and uniform crossover made BlendX the crossover type chosen for this study. The fitness for each test was calculated as follows:

$$Fitness = \frac{abs(Range - DesiredRange)}{DesiredRange} \quad Fitness = \frac{abs(Weight - DesiredWeight)}{DesiredWeight} \tag{3}$$

The next parameters studied were the mutation operators. Both mutation rate and mutation amount were tested in an attempt to find a trade off between quick convergence and not converging to a local minimum. Four different sets of mutation operators were tested using the single stage solid missile system design code matching a range of 400,000 ft and a weight of 2500 lbm. All GA parameters were held constant except the mutation rate and mutation amount. Table 4 shows the different sets of mutation operators and their converged fitness.

Of the cases explored, the best converged fitness was achieved using a mutation rate of 0.20 and a mutation amount of 0.1. This means that the best convergence was obtained by only mutating 20% of the genes in each individual, but those 20% were mutated 10% of the most amount possible. Fig. 14 shows the convergence data for each of the four cases tested.

The last parameter studied was the population size. This parameter is able to change the amount of selection pressure on good members of the population. The fewer the number of members in a population the higher probability that two members with desirable performance could be chosen for the tournament. This allows for quicker convergence, yet again at the cost of potential convergence on a local minimum.

Table 5 shows the results for the population testing. Of the five tests completed Test_5 having a population size of 30 was able to achieve the best fitness.

The results of the GA convergence testing have shown that using a BlendX crossover with a mutation rate of 0.2, mutation amount of 0.1 and a population of 30 yielded the best results for a single stage solid missile design system with two goals match 400,000 ft range and 2500 lb weight goal. Therefore for the binary versus real GA comparisons these parameters will be used for all cases. Changing each parameter for each different GA run could certainly yield better results, however in and effort to compare performance for this study one set of parameters was chosen for each GA and held constant for each test. It should be noted that the mutation operators, crossover type and population size are much more important for the steady state real GA than for the binary generational GA because the binary generational GA gets its diversity from the shear number of guesses for each generation since every member of the population is replaced by a new member. For the steady state GA the only source of diversity is from the mutation operators and crossover type so changing those parameters can make large effects on the convergence. The most complex problem for this study has been deciding on what values to use for these parameters because the results can vary greatly by simply changing the one parameter. In the future a GA could be written in order to optimize the GA parameters for either specific types of goals or a global set that works well for a wide range of goals.

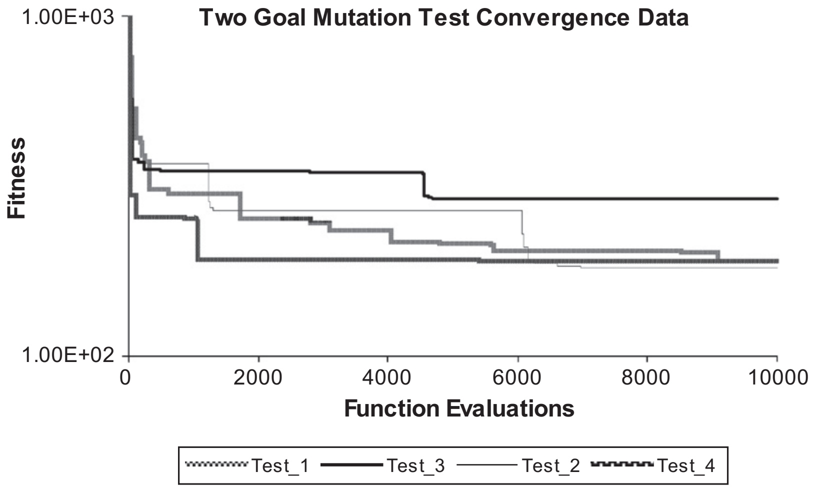


Fig. 14. Single stage solid missile mutation testing 400,000 ft 2500 lb Goal.

Table 5
Population testing fitness comparison.

Population testing	Population size	Fitness
Test_1	3	181.3308951
Test_2	5	172.9612886
Test_3	7	246.3713137
Test_4	10	176.4983529
Test_5	30	169.0051727

5. Missile system designs GA comparisons

Comparisons of the real and binary GA were conducted using all three missile system design codes outlined in Section 3. A wide variety of goals were tested for all three missile system design codes ranging from simple single goal cases such as match range goals to complex two goal cases such as mach range while also matching an initial take off weight. It should also be noted that the difficulty for each GA to find a solution comes from not only the type of goal, but also the limits on the design parameters. In some instances goals were set that were simply not obtainable using the set of design parameters used for these tests.

5.1. Single stage solid missile system design

For both the real and binary GA's, each design parameter is given a maximum, and minimum allowable value. For the binary GA, each parameter is also given a resolution, this resolution is limited by the number of bits as described in Section 1. The initial excitement for the real coded GA came from preliminary comparisons of the real and binary GA's for single stage solid propellant missile design using a single goal, match range. The first two tests conducted demonstrated the real GA's potential for quick and accurate convergence when compared to the binary GA. The convergence plots for both tests are shown in Figs. 15 and 16.

The first preliminary test was to design a single stage solid missile that could hit a target 700,000 ft down range. The real-coded GA was able to achieve a better fitness than the binary GA in just over 2,000 function evaluations, and when let run to the full 10,000 function evaluations was able to converge to a solution 6 orders of magnitude more accurate. It should be noted that for this problem a miss distance of a foot or less is practically considered to be a hit. Therefore the increased precision is for academic purposes only.

The second preliminary test was to design a single stage solid missile that could hit a target 250,000 ft down range. The real GA was able to achieve a better fitness than the binary GA in just over 300 function evaluations, and when let run to the full 10,000 function evaluations was able to converge to a solution 2 orders of magnitude more accurate.

After the initial success of the real coded steady state GA for the first two preliminary tests of the single stage solid code, nine more comparisons were made between the two genetic algorithms using nine different goals. Of the nine tests completed by each GA the steady state real GA was able to converge to a better fitness than the binary generational GA in 6 of the 9 tests. All of the converged fitness's for the goals that the binary GA won were significantly higher than the fitness's

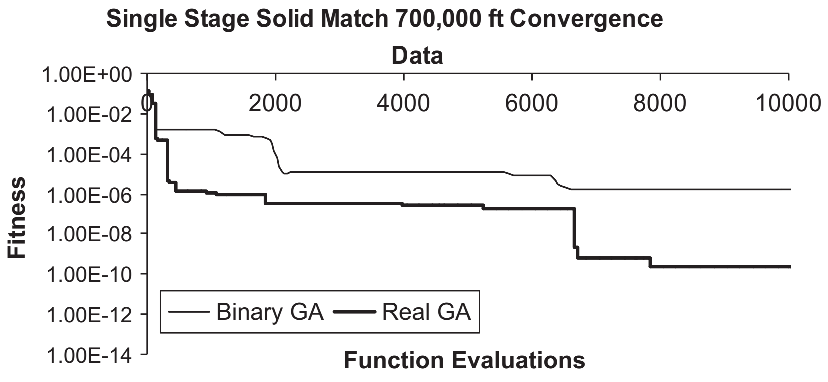


Fig. 15. Single stage solid 700,000 ft test convergence history.

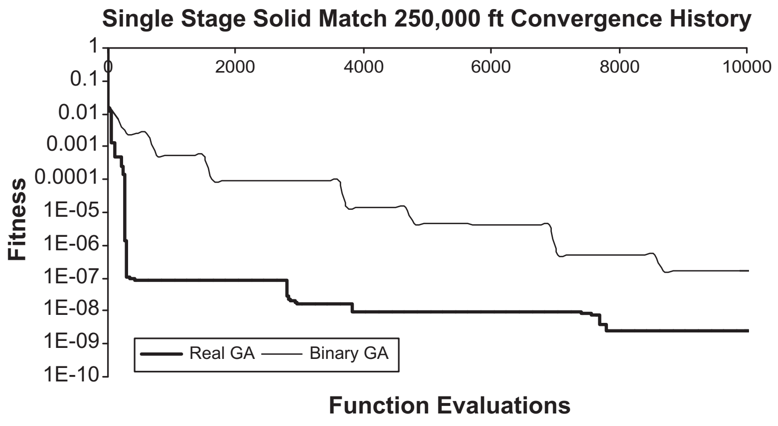


Fig. 16. Single stage solid 250,000 ft test convergence history.

Table 6
Single stage solid testing results.

Single stage solid			
GA fitness comparison	Real GA	Binary GA	F.E.
Match 100 kft Range 2500 lb Weight	50.61	46.69	10,000
Match 350 kft Range 4000 lb Weight	48.67	3.98	10,000
Match 2500 lb Weight 240 s TOF	16.65	16.61	10,000
Match 100 kft Range 10 s TOB	3.06E−15	6.89E−05	10,000
Match 2500 klb Weight 60 s TOF	1.88E−05	3.81E−05	10,000
Match 100 kft Range 15 kft Apogee	3.67E−06	3.58E−04	10,000
Match 20 s TOF	1.25E−05	1.00E−04	10,000
Match 90 kft Range	5.09E−11	9.85E−09	10,000
Match 10 s TOB	3.06E−15	6.89E−05	10,000

that the real steady state GA won as shown in Table 6. The high fitness's demonstrate that for the design parameters specified the missile code was not able to achieve the goals. On the other hand, the tests that the steady state real GA won all have very low converged fitness's. This shows that the missile code was able to achieve those goals using the given inputs. Intuitively this result makes sense because the binary generational GA has more diversity than the real steady state GA, the diversity helps when trying to find the best solution for an unobtainable goal, however that increased diversity results in wasted guesses for more easily obtainable goals, for these goals the real steady state GA is able rapidly home in on a good solution because it is not wasting guesses.

In order to better understand the driving factors in the optimization of the single stage solid missile design code the design parameters were tracked throughout the GA run for both GA's. For this study the match 350,000 ft range, 4,000 lb weight goal is shown to gain a better understanding of how the design parameters are converging. For this test the binary coded genetic algorithm was able to produce a missile that could hit a target 350,000.05 ft down range with a 4,039.75 lb initial take-off weight using 10,000 function evaluations with a converged solution having a fitness of 3.982. This corresponded to a miss distance of 0.05 ft and a miss weight of 39.75 lb. Given the same inputs the real coded GA was able to achieve a fitness of 48.67 after 10,000 function evaluations. This corresponded to a miss distance of 0.00 ft and a miss weight of 485.6 lb.

Fig. 17 shows how both GA's changed the missile body diameter as they converged to their final solution. The binary GA was able to quickly converge to a small missile body diameter, this contributed to its quicker convergence compared to the real coded GA. The real GA's initial guess for body diameter was much larger than the binary GA's initial guess. The initial body diameter for the real GA was 1.95 ft, while the binary GA's initial diameter was 1.43 ft. After 10,000 function evaluations the real GA converged to a body diameter of 1.72 ft, while the binary GA's diameter remained at 1.43 ft.

It should be noted that the overall convergence of the real GA closely follows the minimizing of the missile body diameter, while the binary GA's overall convergence does not. Fig. 18 shows the convergence of another GA design variable, initial launch angle. Fig. 18 shows that the convergence of this design variable contributed more for the overall convergence of the binary GA. Three dimensional models generated by both the real and binary GA's for the match 350,000 ft range 4,000 lb weight test are shown in Figs. 19 and 20.

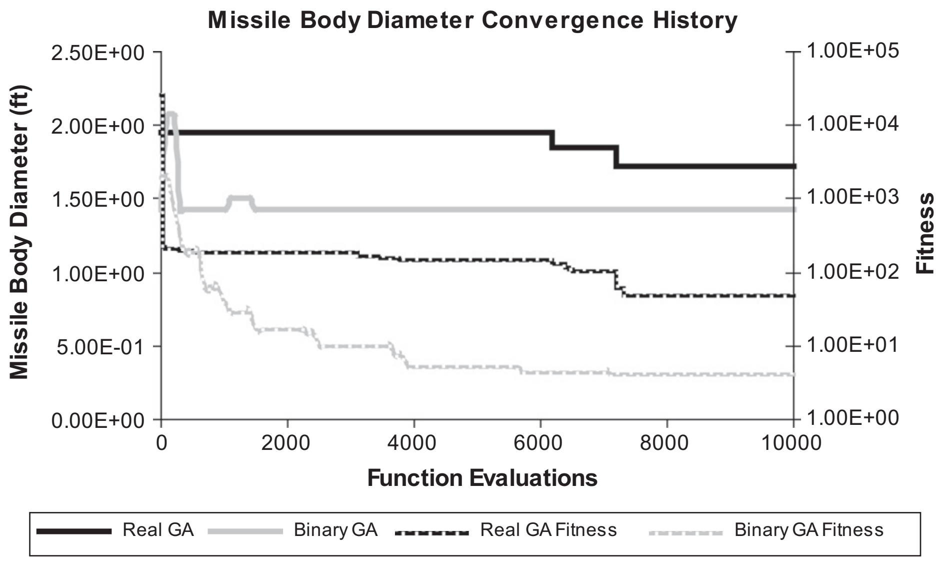


Fig. 17. Single stage solid missile body diameter convergence history.

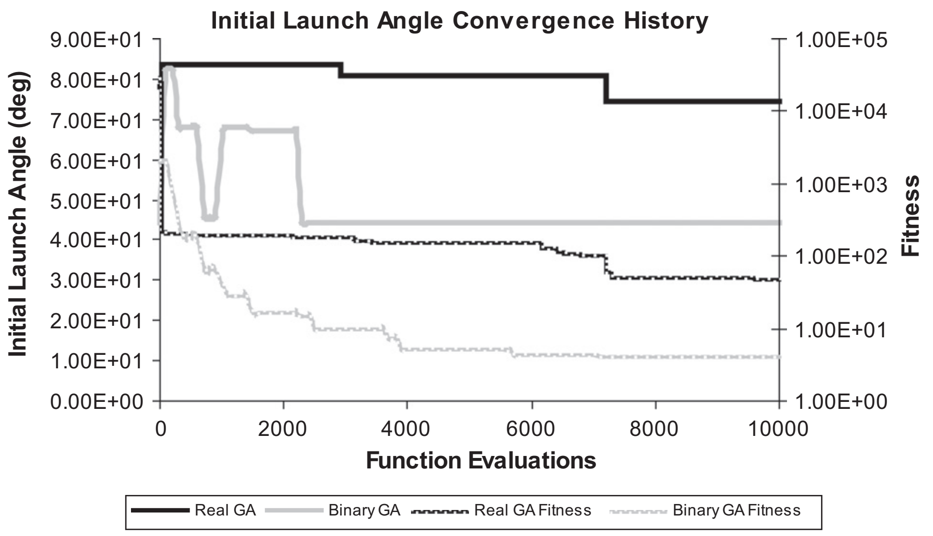


Fig. 18. Single stage solid initial launch angle convergence history.

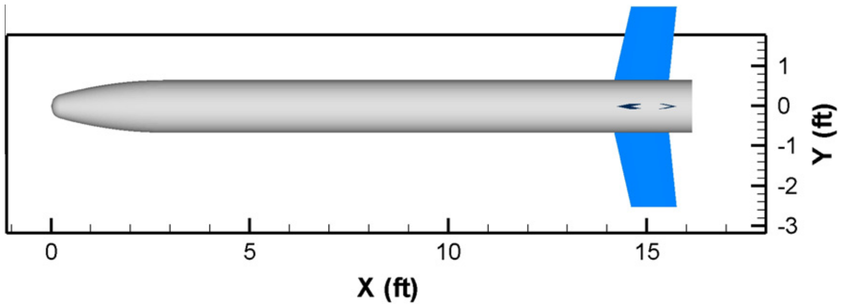


Fig. 19. Single stage solid 3D model-real GA.

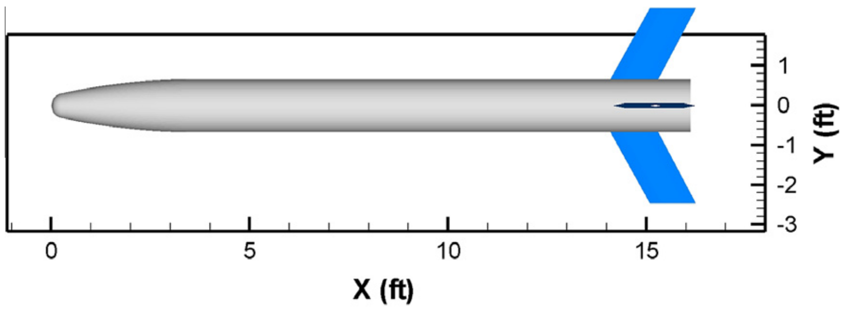


Fig. 20. Single stage solid 3D model-binary GA.

5.2. Two stage solid missile system design

Due to the extended run times associated with the two stage solid missile design code (sometimes in excess of a week) only six goals were used in order to compare the binary generational and real steady stage GA's. The results from the six two stage solid tests are shown in Table 7. Of the six cases tested the real GA was able to converge to a better fitness in four of the six tests. Similarly to the single stage solid tests, for the obtainable goals where fitness's of less than 1.0 could be achieved the steady state real coded GA was able to achieve a better converged fitness. This was the case for two of the seven tests, match 1,056,000 ft range and match 2,112,000 ft range. Because these tests only utilized one goal there are many possible missiles that could be designed to achieve that goal, therefore not as much diversity is needed and the steady state real coded GA can quickly find a set of design parameters that can get close to the range and then make small adjustments without wasting function evaluations in order to match the range. The remaining tests all utilized a second goal in order to limit the number of possible missiles that could be designed to obtain the goal. For the more complex goals the steady state real coded GA was able to converge to a better fitness in two of the four tests, demonstrating that even without the large population size of the generational GA, the steady state real coded GA can get enough diversity by crossover and mutation to out perform the binary generational GA.

In order to gain a better understanding of how each GA was able to converge to their final fitness the design parameters were tracked throughout each GA run as they were for the single stage solid missile design optimization. Only one test was chosen to be discussed for the two stage solid missile design. The test that was chosen was the match 100,000 ft range 6,000 lb initial take-off weight. For this test the binary coded genetic algorithm was able to produce a missile that could hit a target 101,387.69 ft down range with a 8,758.00 lb initial take-off weight using 10,000 function evaluations with a converged solution having a fitness of 414.1. This corresponded to a miss distance of 1,387.69 ft and a miss weight of

Table 7
Two stage solid testing results.

Two stage solid			
GA fitness comparison	Real GA	Binary GA	F.E.
Match 100 kft Range 6 klb Weight	408.59	414.14	10,000
Match 250 kft Range 7 klb Weight	5148.28	314.14	10,000
Match 1056 kft Range	1.14E−02	9.962E−02	10,000
Match 2112 ft Range	1.23E−04	2.67E−01	10,000
Match 528 kft Range min Weight	8.07E+04	7.82E+04	10,000
Match 100 k Range 60 k APO	6.08E+03	6.11E+03	10,000

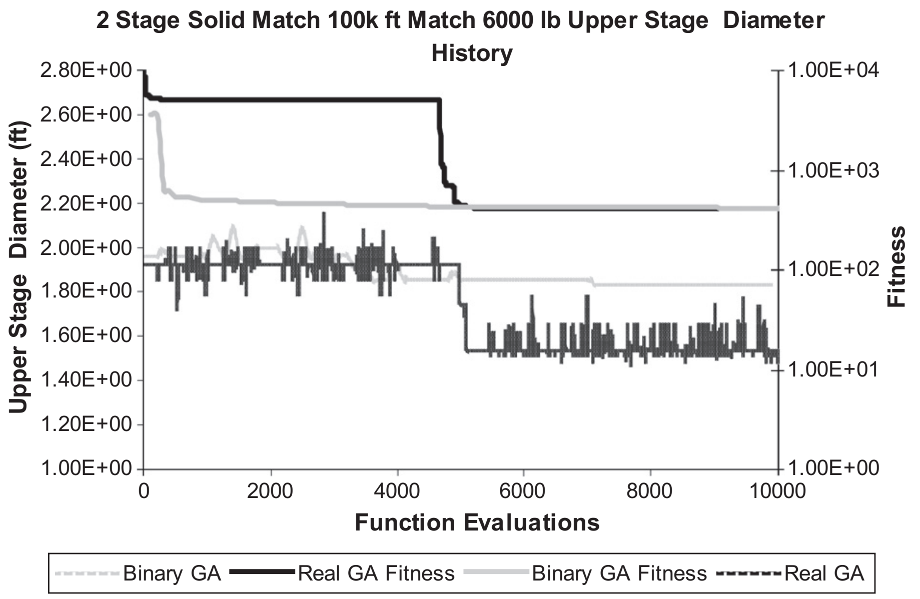


Fig. 21. Two stage solid missile upper body diameter convergence history.

2,758.00 lb. Given the same inputs the real coded GA was able to achieve a fitness of 408.6 after 10,000 function evaluations. This corresponded to a miss distance of 1,427.14 ft and a miss weight of 2,658.80 lb.

The real coded GA was able to produce a slightly more accurate converged solution than its binary counterpart. Fig. 21 shows the how both GA's changed the upper stage missile body diameter as they converged to their final solution. The body diameter is very important when trying to minimize the weight of a missile. For the two stage solid system the body diameter is the only dimensional parameter, all of the other parameters are non-dimensional therefore it is easy to see the overall scale of the missile when looking at the body diameter. For this test the upper stage body diameter could vary between 2.62 and 1.31 ft. The binary GA started with an initial upper stage body diameter of 1.96 ft and converged to a body diameter if 1.83 ft after 10,000 function evaluations. When comparing the overall convergence plot with the body diameter plot there does not seem to be a correlation between the body diameter and the overall convergence for the binary GA. There does seem to be a correlation between the overall convergence and the body diameter for the real GA. After approximately 5000 function evaluations the real GA's body diameter was reduced from 1.92 ft to 1.54 ft, this reduced the weight of the missile and helped to reach a better solution.

Another very important design parameter is the initial launch angle. The launch angle is very important when trying to match a range for a ballistic trajectory. Fig. 22 shows the convergence of the initial launch angle for both GA's. When comparing the overall convergence plot with the launch angle convergence plot it is clear that for both the real and binary GA's the launch angle was a large contributor to the overall convergence. The binary GA started off with an initial launch angle of 63.9 degrees and rapidly converged to a launch angle of 45 degrees within the first 1000 function evaluations. The real GA started with an initial launch angle of 85 degrees and fluctuated slightly for the first 5000 function evaluations and then rapidly decreased to 45 degrees after 5000 function evaluations. The max value allowed for the launch angle was 85 degrees and the minimum value was 45 degrees, therefore better fitness could possibly have been obtained with a broader range of initial launch angles. The 3D models of the two stage solid designed by the real and binary coded GA's are shown in Figs. 23 and 24. After analyzing the 3D models both two stage missiles converged to similar geometries except for the first stage tail sweep. The real coded GA converged to an un-swept tail while the binary GA designed a swept first stage tail. This did not affect the performance of either missile because the sweep of the tail does not play as important of a role in the first stage due to the lower flight speeds.

5.3. Single stage liquid missile system design

A summary of the single stage liquid optimization results is shown in Table 8. For the single stage liquid missile design code the real coded GA was able to converge to a better solution in three of the nine tests. The binary GA was able to converge to better solutions in the remaining test however the fitnesses were on the same order of magnitude. All of the single stage liquid tests were goals that were difficult to obtain using the limits for the 26 design parameters. This is reflected in the fact that the binary generational GA was able to converge to a better fitness than the real steady state GA in six of the nine tests because of its increased diversity due to the large population size and generational operation.

To gain a better understanding of the design parameter convergence for the liquid system, a similar system was set up in order to track the variables throughout the optimization. The test that was chosen was the match 700,000 ft range, 55,000 lb

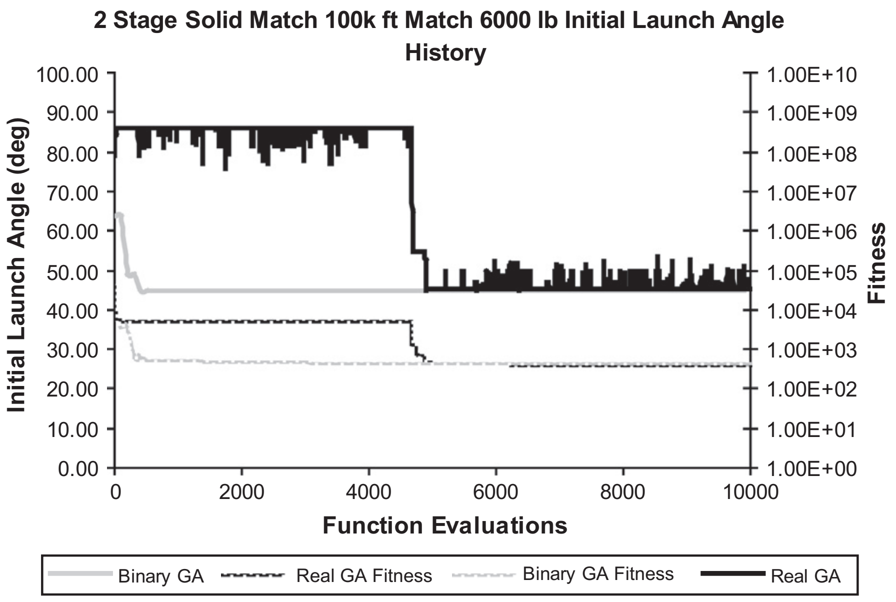


Fig. 22. Two stage solid missile initial launch angle convergence history.

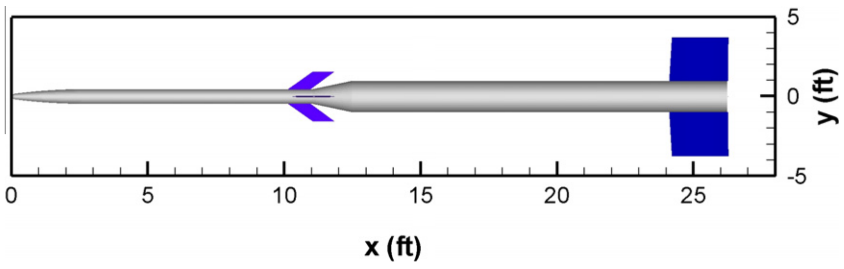


Fig. 23. Two stage solid 3D model-real GA.

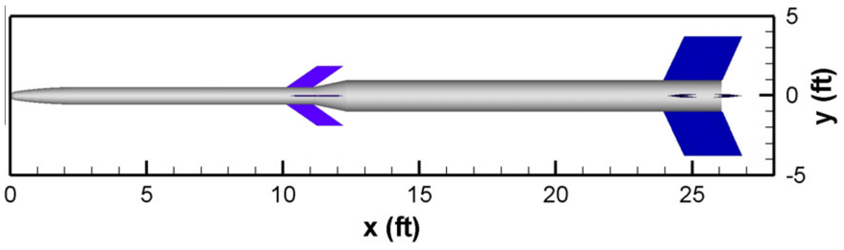


Fig. 24. Two stage solid 3D model-binary GA.

initial take-off weight. The final converged design parameters for each GA are shown in Table 9. For this test the binary coded genetic algorithm was able to produce a missile that could hit a target 700,417.41 ft down range with an initial take-off weight of 55,302.29 lbs using 10,000 function evaluations with a converged solution having a fitness of 72.03. This corresponded to a miss distance of 417.41 ft and a miss weight of 302.29 lbs. Given the same inputs the real coded GA was able to achieve a fitness of 94.00 after 10,000 function evaluations. This corresponded to a miss distance of 848.21 ft and a miss weight of 91.80 lbs. A comparison of the performance of both GA's is shown in Fig. 25.

The binary coded GA was able to produce a slightly more accurate converged solution than its real counterpart. The real GA was able to converge to a lower weight than the binary GA, however it missed the target by a greater distance, therefore the binary GA converged to a lower overall fitness. The real GA's ability to converge to a lower weight can be accounted for by its body diameter convergence as shown in Fig. 25. The limits allowed for the body diameter were a maximum value of 6.6 ft with a 5.0 ft minimum. The real GA started off with an initial body diameter of 6.05 ft and quickly converged to the minimum allowable value for the diameter of 5.0 ft, in order to minimize the weight. The binary GA also started off with a body

Table 8
Single stage liquid testing results.

Single stage liquid			
GA fitness comparison	Real GA	Binary GA	F.E.
Match 450 kft Range 55 klb Weight	6.97	276.16	10,000
Match 700 kft Range 55 klb Weight	94.00	72.03	10,000
Match 450 k ft Range 70 s TOB	58.33	37.04	10,000
Match Range 700 kft	80.63	34.26	10,000
Match 1,056 kft Range	110.43	141.22	10,000
Match 450 kft Range	9.84	1.83	10,000
Match 45 klb Weight 80 kft ALTBO	8000.10	8002.38	10,000
Match 60 s TOB 70 kft ALTBO	4.07	2.92	10,000
Match 700 kft Range 200 s TOF	58.33	37.04	10,000

Table 9
Single stage liquid converged design variables.

Real GA	Binary GA	Design variable definition	
5.4645	5.0021	1	Body diameter ft
4.0000	4.0936	2	Propellant type
0.6000	0.6998	3	Equivalence ratio
2018.8976	2496.5987	4	Chamber pressure psi
0.7143	0.5000	5	Nose dia ratio
1.7333	1.9850	6	Nose length ratio
0.1173	0.1159	7	Nozzle throat dia/dbody
8.5294	21.2879	8	Nozzle expansion ratio
0.8200	0.6176	9	Fractional nozzle length
73.7795	73.8050	10	Burn time sec
0.0333	0.0316	11	Wing root chord ratio
0.8900	0.9188	12	Wing taper ratio
0.0500	0.0322	13	Wing b/2 ratio
2.1429	4.2167	14	Wing le angle deg
0.2206	0.3488	15	XLEwing/lbody
1.0000	1.0410	16	Tail root chord ratio
0.8048	0.5000	17	Tail taper ratio
1.0667	1.0026	18	Tail b/2 ratio
17.0323	10.6517	19	Tail le angle deg
0.9643	0.9763	20	Tail x loc
5000.0000	4999.3174	21	Autopilot delay time sec
0.7890	0.4717	22	Autopilot time const
0.8318	0.7489	23	Autopilot damping coeff
53.5484	52.4772	24	Cross freq Hz
6.2258	5.3116	25	Pronav gain
82.1936	78.0266	26	Initial launch angle deg

diameter above 6.0 ft, however the binary GA's body diameter only converged to 5.50 ft. Both GA's body diameters were converged after only a few thousand function evaluations. The binary GA's overall fitness followed the body diameter convergence, while the real GA on the other hand continued to converge well after the body diameter had converged to the minimum allowable value.

The second design variable shown to play an important role in the convergence of this case was the burn time. A plot of the burn time versus function evaluations is shown in Fig. 26. The burn time was allowed to vary between 150 and 70 s for the liquid missile systems. The initial burn time for both GA's was over 80 s. The binary GA converged to a burn time of 73.8 s in just over 2,000 function evaluations. The real GA converged to the same burn time, however it took just over 7,000 function evaluations. Three dimensional models of the single stage solid designed by the real and binary coded GA's are shown in Figs. 27 and 28. After closely examining the design parameters as well as the actual geometry created for each missile it is clear that since both GA's were able to converge to similar solutions using different missiles that the ultimate limits are not being tested. Because both missiles were able to match similar values for the two goals of match range and weight with distinctly different body diameters the goals must not have been extreme enough to drive the GA's to a unique solution.

6. Conclusions and recommendations

A real coded genetic algorithm has been written from first principles. The real GA operators have been tested to find an optimal set of parameters for quick convergence with minimal convergence on local minimums. The real coded GA was then coupled with three different design codes, a single stage solid missile system design code, two stage solid missile system

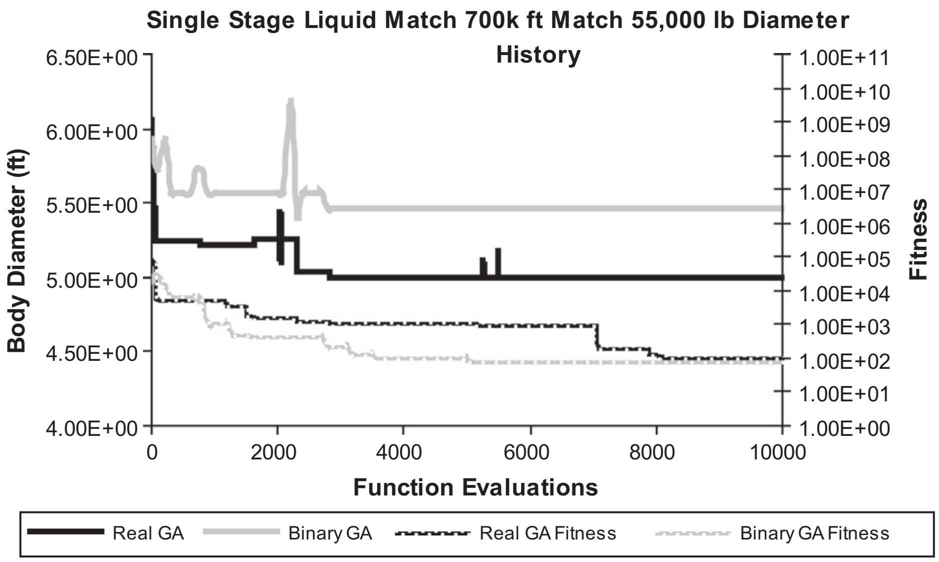


Fig. 25. Single stage liquid missile body diameter convergence history.

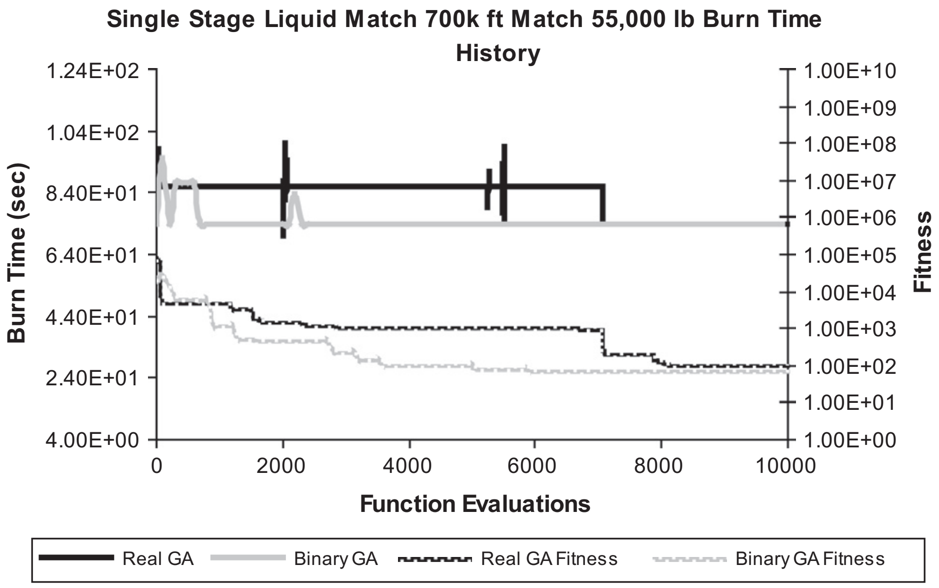


Fig. 26. Single stage liquid burn time convergence history.

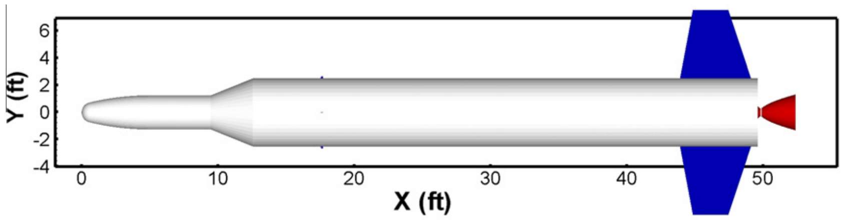


Fig. 27. Single stage liquid 3D model-real GA.

design code and a single stage liquid missile system design code. Twenty-six tests were performed using all three missile system design codes. For each of the twenty-six cases tested both the real and binary GA's were run for 10,000 function evaluations in order to compare the converged fitness values. The real coded GA demonstrated that it is a viable optimization

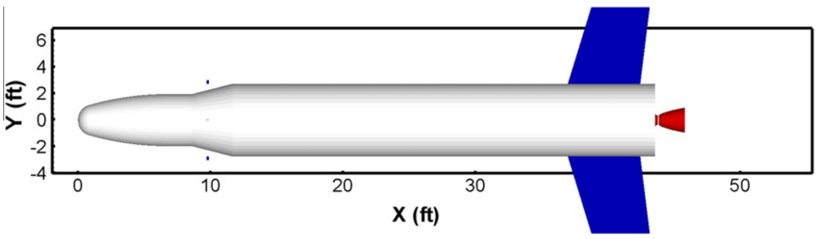


Fig. 28. Single stage liquid 3D model-binary GA.

Table 10
Total testing results.

GA fitness comparison	Real GA	Binary GA
Single stage solid		
Match 100 kft Range 2500 lb Weight	50.61	46.69
Match 350 kft Range 4000 lb Weight	48.67	3.98
Match 2500 lb Weight 240 s TOF	16.65	16.61
Match 100 kft Range 10 s TOB	3.06E−15	6.89E−05
Match 2500 klb Weight 60 s TOF	1.88E−05	3.81E−05
Match 100 kft Range 15 kft Apogee	3.67E−06	3.58E−04
Match 20 s TOF	1.25E−05	1.00E−04
Match 90 kft Range	5.09E−11	9.85E−09
Match 10 s TOB	3.06E−15	6.89E−05
Match 700,000 ft Range	1.21E−10	9.554E−05
Match 250,000 ft Range	1.32E−08	1.544E−06
Two stage solid		
Match 100 kft Range 6 klb Weight	408.59	414.14
Match 250 kft Range 7 klb Weight	5148.28	314.14
Match 1056 kft Range	1.146E−02	9.962E−02
Match 2112 ft Range	1.23E−04	2.67E−01
Match 528 kft Range min Weight	8.07E+04	7.82E+04
Match 100 k Range 60 k APO	6.08E+03	6.11E+03
Single stage liquid		
Match 450,000 ft Range 55,000 lb Weight	6.97	276.16
Match 700,000 ft Range 55,000 lb Weight	94.00	72.03
Match 450,000 ft Range 70 s TOB	58.33	37.04
Match range 700,000 ft	80.63	34.26
Match 1,056,000 ft Range	110.43	141.22
Match 450,000 ft Range	9.84	1.83
Match 45,000 lb Weight 80,000 ft ALTBO	8000.10	8002.38
Match 60 s TOB 70,000 ft ALTBO	4.07	2.92
Match 700,000 ft Range 200 s TOF	58.33	37.04

tool when compared to the already robust binary GA. Of the twenty-six comparative tests, the real GA was able to converge to a better fitness in fifteen tests. Table 10 shows an overview of the final fitness's achieved by both the real and binary GA's for the eleven cases tested.

More research needs to be conducted on the GA parameters themselves to determine either an overall optimum set of GA parameters for the real coded GA, or a way to vary all of the parameters throughout the GA run to account for the differing parameters needed for differing problems. A second GA could be used to optimize the GA operators to get a set of operators that is truly optimized over a wide variety of tests.

The steady state real coded GA was able to beat the binary generational GA for goals that are obtainable using the given design parameters. For these tests the real coded GA was able to very rapidly converge to much more accurate fitness than the binary GA. This is due to the fact that the steady state GA did not have to waste function evaluations on bad members. The steady state real GA has proven that it is very effective at optimizing a system that is fairly well known, however for a very complex unknown system the generational binary GA with its large population size and increased diversity is better. While the binary GA converged to better fitness values for the many of the hard to obtain goals conducted for this study, the real GA can still be much more effective than the binary GA for more complex problems involving 60 or more design variables because of the bit limitation inherent with the binary GA.

The real GA is inherently better than the binary GA for more complex optimizations because the real coded GA has no limit on resolution other than what a double precision real number can handle. Also since the real-coded GA does not have to convert real numbers into a single binary bit string it is not susceptible to hamming cliffs therefore making more complex optimizations with more design parameters and finer resolutions possible. Because the real-coded GA is operating directly on the design parameters rather than bits of design parameters it is much easier to gain an understanding of how the GA is modifying the parameters. This could allow the user to track which parameters are affecting the convergence the most and select them for more or less mutation throughout the optimization to increase the convergence. This would be much more difficult for the binary GA because all of the design parameters are represented by a single bit string.

In the future a real coded generational GA could be written and coupled with the real coded steady state GA. The generational GA could be used at the beginning of the optimization, and after a set number of generations the GA could be switched to steady state to home in on a very accurate fitness, therefore using the best characteristics of both types of genetic algorithms. The GA parameters could also be adjusted on the fly throughout the GA run in order to yield quicker more accurate convergence. Other types of population based optimizers should also be researched such as particle swarm optimization (PSO) and compared to both binary and real genetic algorithms for aerospace optimization applications.

References

[1] C.L. Karr, L.M. Freeman, D.L. Meredith, Genetic algorithm based fuzzy control of spacecraft autonomous rendezvous, NASA marshall space flight center, in: Fifth Conference on Artificial Intelligence for Space Applications, 1990.
[2] K. Krishnakumar, D.E. Goldberg, Control system optimization using genetic algorithms, Journal of Guidance, Control, and Dynamics 15 (3) (1992).
[3] G. Torella, L. Blasi, The optimization of gas turbine engine design by genetic algorithms", AIAA Paper 2000-3710, in: 36th AIAA/ASME/SAE/ASEE Joint Propulsion Conference and Exhibit, July 2000.
[4] M.G. Perhinschi, A modified genetic algorithm for the design of autonomous helicopter control system, AIAA-97-3630, in: Presented at the AIAA Guidance, Navigation, and Control Conference, New Orleans, LA, August 1997.
[5] S. Mondoloni, A genetic algorithm for determining optimal flight trajectories", AIAA Paper 98-4476, in: AIAA Guidance, Navigation, and Control Conference and Exhibit, August 1998.
[6] M.B. Anderson, Using pareto genetic algorithms for preliminary subsonic wing design", AIAA Paper 96-4023, in: Presented at the 6th AIAA/NASA/USAF Multidisciplinary Analysis and Optimization Symposium, Bellevue, WA, September 1996.
[7] R.E. Perez, J. Chung, K. Behdinan, Aircraft conceptual design using genetic algorithms", AIAA Paper 2000-4938, in: Presented at the 8th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, September 2000.
[8] M.B. Anderson, J.E. Burkhalter, R.M. Jenkins, Design of an air to air interceptor using genetic algorithms, AIAA Paper 99-4081, in: Presented at the 1999 AIAA Guidance, Navigation, and Control Conference, Portland, OR, August 1999.
[9] M.B. Anderson, J.E. Burkhalter, R.M. Jenkins, Intelligent systems approach to designing an interceptor to defeat highly maneuverable targets", AIAA Paper 2001-1123, in: Presented at the 39th Aerospace Sciences Meeting and Exhibit, Reno, NV, January 2001.
[10] J.E. Burkhalter, R.M. Jenkins, R.J. Hartfield, M.B. Anderson, G.A. Sanders, Missile systems design optimization using genetic algorithms, AIAA Paper 2002-5173, in: Classified Missile Systems Conference, Monterey, CA, November, 2002.
[11] Roy J. Hartfield, Rhonald M. Jenkins, John E. Burkhalter, Ramjet powered missile design using a genetic algorithm, AIAA 2004-0451, in: Presented at the forty-second AIAA Aerospace Sciences Meeting, Reno NV, January 5–8, 2004.
[12] R.M. Jenkins, Roy J. Hartfield, John E. Burkhalter, Optimizing a solid rocket motor boosted ramjet powered missile using a genetic algorithm", in: AIAA 2005-3507 presented at the Forty First AIAA/ASME/SAE/ASEE Joint Propulsion Conference, Tucson, AZ, July 10–13, 2005.
[13] Burger, Christoph, Roy J. Hartfield, Propeller performance optimization using vortex lattice theory and a genetic algorithm", AIAA-2006-1067, in: Presented at the Forty-Fourth Aerospace Sciences Meeting and Exhibit, Reno, NV, Jan 9–12, 2006.
[14] B. Chernyavsky, V. Stepanov, K. Rasheed, M. Blaize, D. Knight, 3-D hypersonic inlet optimization using a genetic algorithm", AIAA Paper 98-3582, in: 34th AIAA/ASME/SAE/ASEE Joint Propulsion Conference and Exhibit, July 1998.
[15] S.Vukovic, L. Sopta, Binary coded and real coded genetic algorithm in pipeline flow", AMS Paper 35–42, in: 7th International Applied Mathematical Communications Conference, 1998.
[16] L. Elliot, D.B. Ingham, K.G. Kyne, A Real coded genetic algorithm for the optimization of reaction rate parameters for chemical kinetic modelling in a perfectly stirred reactor, in: Proceeding of Genetic and Evolutionary Computational Conference, New York, 2002.
[17] S.V. Sugala, P.K. Bhattacharaya, Real coded genetic algorithm for optimization of pervaporation process parameters for removal of volatile organics from water, Industrial Engineering and Chemistry Research 42 (13) (2003) 3118–3128.
[18] Holland, Adaptation in natural and artificial systems, The MIT Press; Reprint edition 1992 (originally published in 1975).
[19] M.B. Anderson, Users Manual for IMPROVE© Version 3.0", Sverdrup Technology Inc./TEAS Group.
[20] G. Dozier, H. Cunningham, W. Britt, F. Zhang, Distributed constraint satisfaction, restricted recombination, and hybrid genetic search, in: The Proceedings of the 2004 Genetic and Evolutionary Computation Conference (GECCO-2004), LNCS, Springer, Seattle, WA, 2004, pp. 1078–1087.
[21] E. Unsal, J.H. Dane, G.V. Dozier, A genetic algorithm for predicting pore geometry based on air permeability measurements, The Vadose Zone Journal 4 (2005) 389–397. Soil Science Society of America, Madison, WI.
[22] G. Dozier, A. Homaifar, E. Tunstel, D. Battle, An introduction to evolutionary computation, in: Intelligent Control Systems Using Soft Computing Methodologies, A. Zilouchian, M. Jamshidi (Eds.), CRC press, pp. 365–380, (Chapter 17).
[23] J.N. Siddal, Optimal Engineering Design: Principles and Applications, Mercell Dekker Inc., New York, NY, 1982.
[24] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, Introduction to algorithms, Second Edition. MIT Press and McGraw-Hill, 2001. ISBN 0-262-03293-7. Section 29.3: The simplex algorithm, pp. 790–804.
[25] L.J. Eshelman, J.D. Schaffer, Foundations of genetic algorithms 2, Real Coded Genetic Algorithms and Interval Schemata, 5–17, San Mateo, CA, 1992.
[26] L.J. Eshelman, J.D. Schaffer, Real-coded genetic algorithms and interval-schemata, in: Foundations of Genetic Algorithms-2, Morgan Kaufman, San Mateo, CA, 1993, pp. 187–202.
[27] David B. Fogel, Evolutionary Computation: Toward a New Philosophy of Machine Intelligence, IEEE Press, 1995.
[28] F. Vavak, T. Fogarty, Comparison of Steady State and Generational Genetic Algorithms for Use in Non-Stationary Environments, IEEE Press, 1996.
[29] D.K. Huzel, D.H. Huang, Modern engineering for design of liquid-propellant rocket engines, American Institute of Aeronautics and Astronautics 147 (1992). Washington, DC.