Genetic Algorithm Optimization of Inlet Bleed Design for a Hypersonic Jet Engine with Mode Transition

Kyle B Gaiser*
Case Western Reserve University, Cleveland, Ohio, 44106

Meng-Sing Liou†
NASA Glenn Research Center, Cleveland, Ohio 44135

A genetic algorithm coupled with computational fluid dynamic software is used to optimize the configuration of an engine inlet at a supersonic speed. The optimization program is written to calculate the pressure recovery of many varying bleed schedules throughout the inlet walls. The goal is to find the best combination of the bleed holes' locations, diameters, and flow rates such that a high pressure recovery is maintained. Parallel computing using the NAS supercomputer Columbia is used to run the algorithm efficiently. This is the first time a genetic algorithm has been applied to inlet bleed design. A test function is used to evaluate and debug the optimization algorithm. The genetic algorithm and its associated programs are found to work and show strong potential for use in developing more efficient bleed schedules in a hypersonic engine.

Nomenclature

r_d	=	pressure recovery
p_{0a}	=	atmospheric stagnation pressure
p_{02}	=	upstream inlet stagnation pressure
p	=	pressure
p_t	=	total pressure
ρ*	=	non-dimensionalized density
u*	=	non-dimensionalized velocity in the direction of freestream flow
w*	=	non-dimensionalized velocity in the direction perpendicular to jet's floor
v*	=	velocity in the direction perpendicular to the inlet's sidewall
e_0*	=	non-dimensionalized energy
γ	=	ratio of specific heats
Re	=	Reynolds number
M	=	Mach number
dimin	=	the diameter's precision; controls how many decimal places to which the diameter is carried out
spacemin	=	controls the number of decimal places to which the hole location is carried out
dimin	=	the minimum diameter allowable for a hole
sumfitness	=	sum of the individuals' fitness values in a given generation
pmutation	=	probability of mutation for each parameter
pcross	=	probability of crossover for each parameter
edge	=	buffer distance that accounts for edge effects near holes

Background

A. Genetic Algorithms

* Research Associate, 2007 NASA Glenn Academy, 1623 E. 115th St, Cleveland, OH, 44106, kbg4@case.edu.
† Project Mentor and Research Scientist, Turbomachinery and Propulsion System Division, MS 5-11, 21000 Brookpark Rd, Cleveland, OH, meng-sing.liou-1@nasa.gov, Associate Fellow AIAA.

Genetic Algorithms are a relatively new and powerful optimization technique based upon the fundamentals of biological evolution. There are many optimization techniques in existence, like the gradient search or grid search method. The goal of these techniques is to find the best combination of a set of parameters, such that the best result for a given objective is obtained. In engineering design, these techniques can be used to maximize the efficiency of a system. As need for numerical methods of optimization for complex engine designs increases, the genetic algorithm provides a robust, efficient solution.

How a genetic algorithm works

A biological system contains a population of individuals, each having a unique expression of traits. These traits are the design variables of an engineering system. The expressions of the traits are called alleles in biological terms, or parameters in engineering terms. Each individual is composed of a combination of these parameters, a string, also known as a chromosome. The chromosomes contain the genetic makeup of the individual. In a genetic algorithm, each individual represents a different design configuration.

Some individuals are better fit for their environment than others, depending on the genetic makeup of each individual. In biological evolution, the environmental surroundings determine which individuals are better fit, and natural selection determines which individuals will survive in their environment to pass on their unique set of parameters. In a genetic algorithm, a fitness function expresses the objective in terms of the parameters. The output of the fitness function, also known as the objective function, is a fitness value that is assigned to each individual based upon how well the combination of parameters performed with respect to the design objective.

Crossover is the essential step that carries over the best design variables to the next generation. In nature, survival enables better fit individuals to reproduce, thus passing along their chromosomes to the next generation. Poorly fit individuals die out because their parameter combinations result in poor adaptation to the environment. Similarly, the genetic algorithm randomly selects individuals to crossover, although the better fit designs have a higher probability of being selected for crossover. When two parents crossover, some traits from each are swapped and the remaining traits stay the same. The processes of selection and crossover are described in more detail in the genetic algorithm section below.

Mutation is another operation that works in conjunction with crossover. Instead of swapping certain parts of the genetic makeup, mutation randomly changes a parameter to a new value that is likely not a parameter of either of its parents'. In the algorithm, mutation occurs directly after crossover, on a new individual. Mutation ensures a variety of parameter values, so that the whole design space is considered, not just the parameters within the initial population.

Advantages of the Genetic Algorithm

Among the many numerical optimization methods, the genetic algorithm has many advantages that make it extremely powerful. The genetic algorithm is known as a global optimization method, as opposed to a local search. This means it can effectively search the whole design space and find the true optimal solution, instead of "hill-climbing" like other methods. It does not require any knowledge of what makes a "good" design ahead of time; it is a blind search and can start from any baseline. The genetic algorithm is also multi-variable and multi-objective. Many design parameters can be tested, and furthermore, the parameters can be optimized when more than one objective is considered. It is adaptable to multidisciplinary uses. Within the past few decades it has been tailored for engineering design problems. While the genetic algorithm is more computationally demanding than other optimization techniques, it is ideal for parallel computing, which drastically increases its efficiency.

B. Inlet Bleed Design

The inlet is the upstream engine component that directs the incoming air for combustion downstream. It plays a crucial role in slowing down the air, with minimal losses, so that the air/fuel mixture and combustion has enough time to take place before it is accelerated and exhausted by the nozzle. Moreover, the inlet is responsible for maintaining high stagnation pressure (high pressure recovery) and uniform velocity and direction of flow entering the compressor.

A high pressure recovery across in the inlet is critical to the engine's performance. Friction due to the inlet walls creates non-isentropic flow, slowing down the bordering air known as the boundary layer. Normally, shear forces (viscous and turbulent) keep the boundary layer from peeling off the wall. However, if there is a large, adverse pressure gradient across the inlet, boundary layer separation occurs, reversing flow and creating large pressure drag. This causes a dramatic drop in efficiency and the engine stalls.

In the case of a supersonic jet, the inflow must be slowed down to the subsonic region as it leaves the inlet and enters the turbomachinery. As air is decelerated there is a build up in pressure at the front of the inlet, until a shock

wave is released. The inlet is designed to carry out several oblique shocks with smaller energy losses and a small normal shock at the end of the inlet throat, instead of one large normal shock, which is very energy inefficient. The regions where oblique shocks hit the inlet wall are areas where boundary layer separation is more likely to occur.[1]

Taking into account these losses, a technique called bleeding is used to minimize or eliminate boundary layer separation. Sections of the inlet wall are perforated; holes or slots are deliberately engineered to allow the boundary layer air to escape. Generally, the holes are placed at sections where the oblique shocks strike the inlet wall. Flow separation is prevented by removing the low momentum air where shock is located, the area where separation would normally occur.

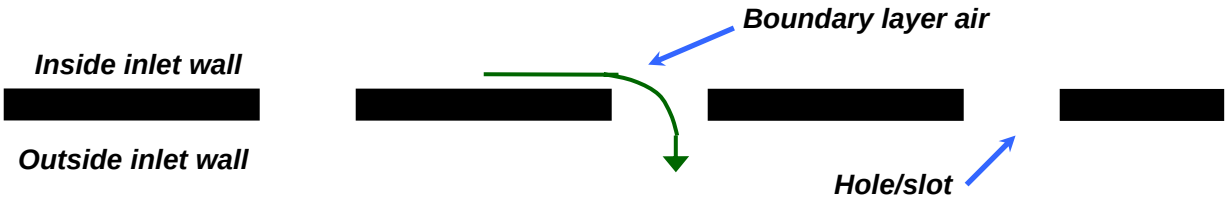

Figure 1. Model of Bleeding. *Inflow air is bled through perforated inlet walls*

C. Hypersonic Jet Engine

Hypersonic engines are a promising field of research for air and space vehicles. They will not only be the fastest aircraft, they will be versatile, transitioning between sub, super, and hypersonic modes of flight. A single aircraft will be capable of the broadest array of speeds. When fully developed, hypersonic vehicles have the potential to become a single-stage escort to and from space.

Traditionally, inlets are designed for specific flight speeds. Designing a wide range velocity inlet to eliminate boundary layer separation requires innovative research and engineering techniques, such as a genetic algorithm.

The Design Problem

The optimization problem considers the design of an inlet with mode transition. The hypersonic jet has two different inlets separated by a pivoting cowl (Fig. 2). The low speed inlet is the turbojet flow path, which is used for flight speeds ranging from mach 0 to 4. The high speed inlet is for the scramjet for speed above mach 4. The optimization evaluates the turbojet inlet at a flight speed of mach 4 and a Reynolds number of 1.7×10^7.

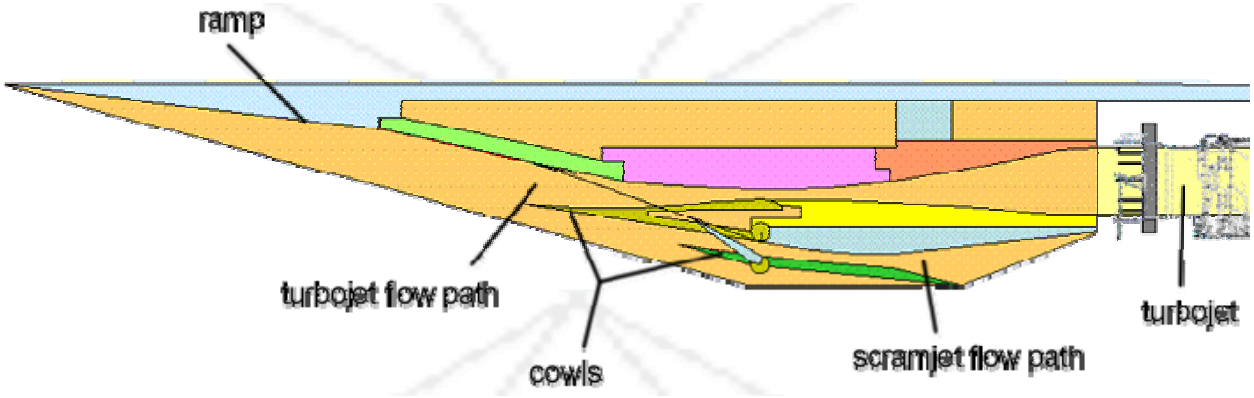

Figure 2. Hypersonic Jet Engine Inlet. *Low speed inlet is on the top, high speed scramjet inlet is on the bottom.*

For the scope of this project, two-dimensional flow is considered. A total number of seven holes (also known as slots) are evaluated in different sections along the top and bottom inlet walls (Fig. 6). The location, the diameter of the holes ("length" if dealing with a slot; however, "length" and "diameter" can be used interchangeably since the z-direction does not matter in a two-dimensional analysis), and the mass flow rate through each hole are design variables. Since there are a total of seven holes and three design variables per hole, the total number of design variables is twenty one. The values of these twenty one variables are the parameters, or alleles, that the genetic algorithm uses to optimize the design.

Dave Saunders, et al[‡] is currently testing different bleed schedules for the hypersonic inlet in the 1x1 wind tunnel located at the Glenn Research Center. The baseline design configurations and constraints for the bleeding sections were provided by his research. The location of the holes is constrained to be within specific sections of bleed in the ramp and cowl (Fig. 6). These are the sections tested in the wind tunnel. The diameter of each hole varies between 0 and 6 inches. A hole with 0 in. diameter means there is no bleeding within the given section, which is a valid possibility within the design space. The maximum diameter is chosen based on the largest section of bleed, which is 6 in. An important difference to note is that the wind tunnel tests many small holes within a section, while the genetic algorithm tests one hole within a section; however, the hole has a diameter range such that it can cover the entire section, which is equivalent to many small holes across the section. More detail on how these constraints were calculated is in the description of the initial population generator program below. The range of the mass flow rate is 0 to .05 for each hole. This means the maximum total bleed could be .35, or 35% of the captured mass flow rate. In this case, each individual design needs only one chromosome. The individual's chromosome is made up of the twenty one parameters.

The fitness function that evaluates the performance of each individual is the pressure recovery. The pressure recovery formula is

$$r_d = p_{02} / p_{0a}$$

The objective is to find the combination of parameters that returns the highest pressure recovery. Since it is a ratio, pressure recovery will range from 0 to 1.

Table 1 Boundary conditions.

M	4
T_{inf} °R	518.7
Re x 10^7 /m	1
γ	1.4
Flow rate	0 - .35
Diameter in.	0 - 6
Location in.	88.5 – 178.898

Approach

A. Tools

Force

Force is the FORTRAN editor and compiler that is used to write all the genetic algorithm and grid generating programs. F-77 is used for the majority of the program, although some aspects of F-90 are used as well.

Overflow

Overflow is Computational fluid dynamic (CFD) software that calculates the flow behavior in the inlet and outputs the results (temperature, density, etc) for each grid point along the inlet. Overflow requires that every grid line for the inlet be assigned a specific boundary condition (i.e. freestream flow, viscous adiabatic wall, jet mass flow condition for a hole) before it runs.

EnSight

This visualization tool plots the inlet grid and the results file from Overflow. Pressure, temperature, density, and much more can be represented graphically in EnSight.

Columbia

The NASA Advanced Supercomputer (NAS) allows for fast parallel processing of many generations in the genetic algorithm. Columbia uses UNIX.

[‡] Aerospace Engineer, Turbomachinery and Propulsion System Division, MS 5-12, 21000 Brookpark Rd, Cleveland, OH, john.d.saunders@nasa.gov

B. Programs

The programs for the genetic algorithm were mostly written from scratch specifically for this design problem. A few functions and programs were adapted for the algorithm. For this reason, each program used to run the optimization is broken down and explained. The three main programs utilized by the optimization are the initial population generator, the grid generator, and the genetic algorithm. A diagram of how these programs interact with each other and the software is shown below in Figure 3. A batch script is written, initially in Bourne shell and then converted to c shell, to link together the programs and software. For a detailed account of how the batch script works, see Appendix I.

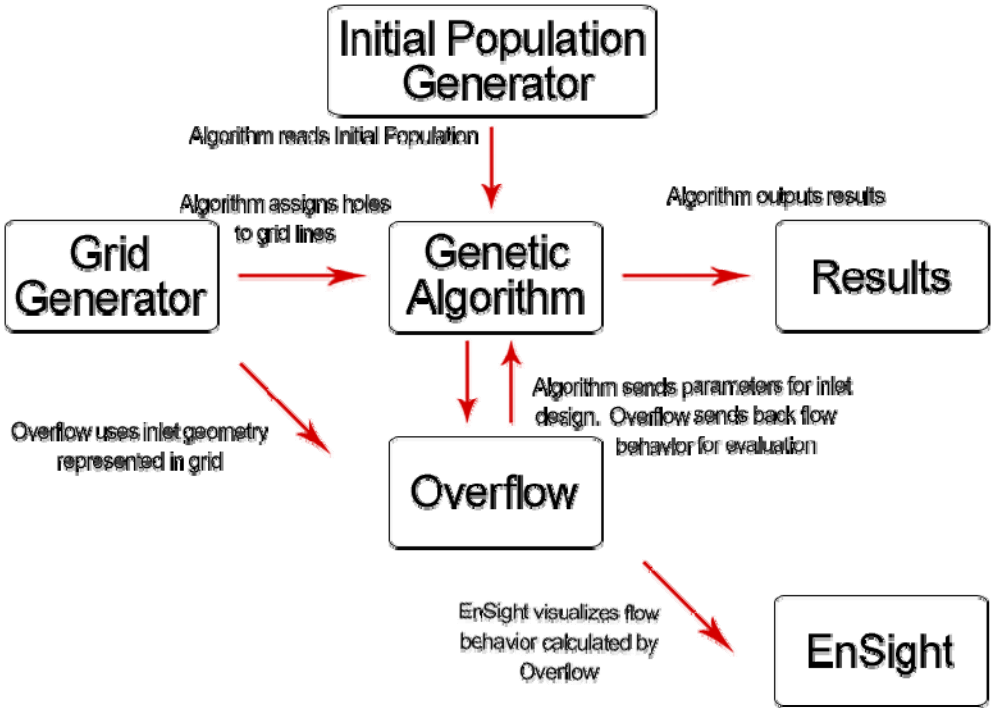

Figure 3. Flowchart of optimization program.
These programs are linked together by a batch script

Grid Generator

The grid generator is the very first program to run in the batch file and it only runs one time. The user defines the desired density of gridlines for a region of holes and a region of space where holes are not located (number of lines per hole or space). Denser i-lines (the vertical grid lines) are needed where sections of bleed occur because this is where the flow is most important. Denser j-lines (horizontal grid lines) are needed near the top and bottom walls, since this is where the boundary layer is located. This means that the highest resolution of gridlines occurs near the wall and at sections of bleed, where understanding the flow is most crucial. The sections of bleed do not take up the entire length of the inlet. In fact, the first section of bleed occurs at x = 108.5 in. and the last section occurs at 159.898 in. The origin is at (0,0) and the total length of the inlet is approximately 297.92 in. It is also important to have dense grid lines a significant distance before and after sections of bleed since bleeding will affect the flow in these areas as well. There is a variable, *edge*, that takes into account these edge effects, and can be easily changed by the user.

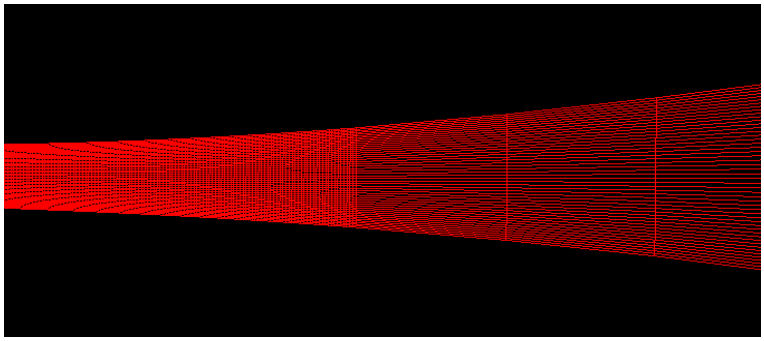

Figure 4. Current grid. *Grid lines permanently set. As holes move, new grid lines assigned to hole. Zoomed in to see fine lines. At x = 159.898 + edge dense lines stop and go to wider spaced lines.*

With these inputs, the grid generator creates a static grid, with a fixed number of

grid lines, each assigned to a specific x-value that does not change. The i-lines are more dense in between x = (108.5 – edge) and x = (159.898 + edge), where bleeding sections are located (Fig. 4). The j-lines use a Gaussian function to a lot more weight and thus higher density, to j-lines that are closer to the walls.

The output is a single Plot3D formatted grid file. For every generation, the hole locations move via crossover and/or mutation. Each run, the genetic algorithm simply calls the grid file and assigns every hole a new set of i-lines.

Originally, the grid generator was written to adapt to the holes as they changed diameter from one individual to another and from one generation to another (Fig. 5). The density of the i-lines would dynamically change as a function of the hole's diameter. This method had a couple problems. First, if two holes were very close to each other, the weighting could prohibit any i-lines from be assigned in between the holes. Secondly, there would need to be a function that creates i-lines outside the holes as a buffer to account for the edge effects described above. Lastly, with holes on the top and the bottom, i-lines could cross and this would create a problem with overflow.

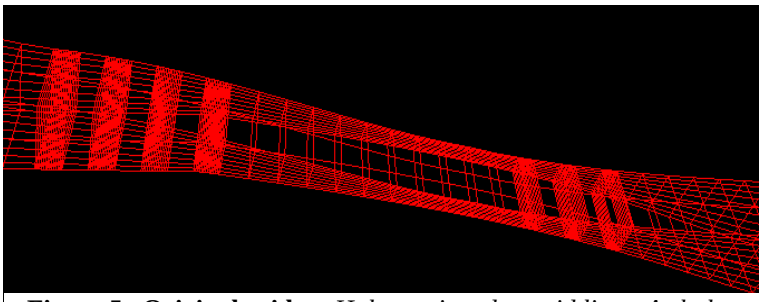

Figure 5. Original grid. *Holes assigned to grid lines. As holes move, grid lines move.*

Another idea considered was to permanently assign grid lines to holes. As holes moved, the grid lines would move, changing their x-values. Again, this created a problem with overlapping grid lines. The current method, the first method described above and shown in Figure 4, allows for overlapping x locations of top and bottom holes without any crossing of grid lines. It also allows for hole diameters to go to zero without assigning multiple i-lines to the same x-value.

The geometry of the inlet was calculated by using a best fit program adapted from Garcia's *Numerical Methods for Physics.*[2] The inlet's x and y coordinates were given by Saunders and these were broken up into manageable sections and inputted into the fitting program. The output was two sets of polynomials (ranging from first order to fourth order) that described the inlet's top and bottom wall. The intersection points of the polynomials in each set were used as the cutoff between one function and the other. The result was two piecewise functions representing the inlet's geometry. It should be noted that since a best fit polynomial was used, not all points are exactly the same as the geometry provided by Saunders, et al. The points are very close to the original baseline and for the scope of this project, the geometry more than suffices.

Initial Population Generator

The initial population generator runs directly after the grid generator and randomly assigns the alleles, or parameter values, of all the individuals in the first generation. A random number generator with a changing seed is used so that the initial populations are not the same each time the optimization runs. The random number generator chooses numbers between 0, inclusive, and 1, exclusive. The location, diameter, and flow rate alleles are randomly generated within the minimum and maximum allowable values using the general form:

```
int(rand1(0)*(max+1-min))+min
```

where max and min are the maximum and minimum values of the constraint, respectively.[3] The location constraint is uniquely programmed. As shown in Figure 6, the first three sections of bleed (RX, R1, and R2) are located upstream from any sections of bleed on the top wall. To test for a larger range of possibilities, the location of the first three (out of seven total) holes can be assigned anywhere within those three sections. For example, three holes can be located in section RX and none in the R1 and R2, or each of the three sections can contain one hole, etc. Two holes are assigned to each top/bottom section pairs. Holes four and six are in R3 and R4, and holes five and seven are in C1 and C2, respectively. Furthermore, subtracting *dimin* from max, ensures that a location is assigned far enough away from the end of a section so that a diameter can still be assigned to that hole without it extending outside the bleed section.

The precision of each value is also programmed and can be changed by the user. The code is written as:

```
real(int(rand1(0)*(max*diprec+1-min*diprec))+min*diprec)/diprec
```

Next, the initial population generator runs a series of checks. The first check is for double location, which makes sure that all locations are far enough apart that a diameter can be assigned between holes. If double location occurs, then the upstream location is reassigned a location value. The second check is for overlapping holes, which ensures that no hole's diameter runs into the next hole's location. The odd and even (top and bottom) holes for sections R3, R4, C1, and C2 are exempt from this check since it is okay to have a top and bottom hole share i-lines. However, any two odd numbered holes or two even numbered holes within those sections must be checked for overlap since they are both on the same wall. In this initial case of only seven holes, this does not occur because there is only one hole on the bottom and one hole on the top for the later two top/bottom paired sections.

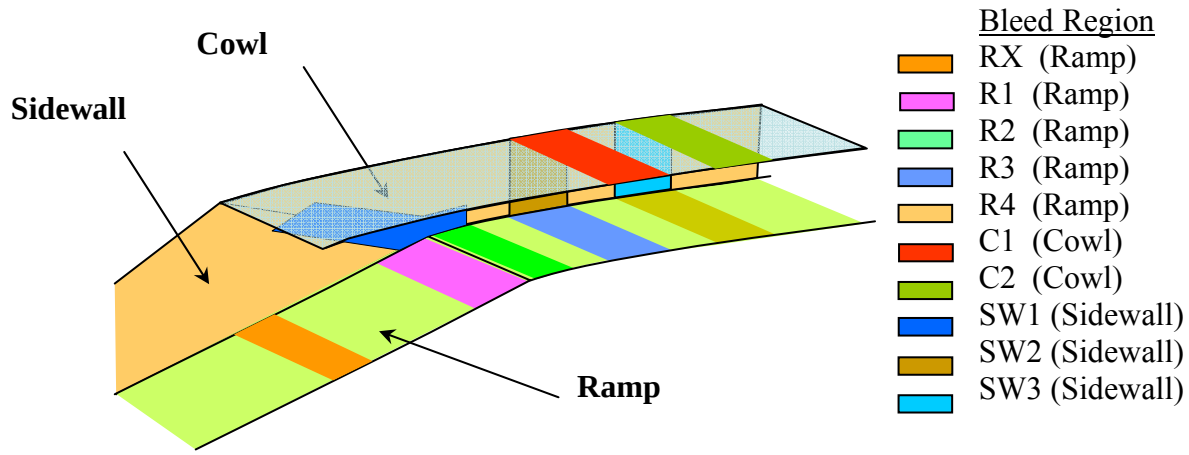

Figure 5. Low speed inlet bleed regions (Mach 4 coordinates).

Once the initial population is created, the location, diameter, and flow rate parameters must be expressed in a format that overflow understands. Since overflow requires the boundary conditions for each individual before it runs, a new input file is written for each. The program imports the grid file created from the grid generator and searches for the i-line whose x-value is closest to the starting and ending location of each hole. These i-lines are given the corresponding boundary condition for a hole, and the mass flow rate is written to each hole as well. This input file is an essential aspect of the initial population generator and the genetic algorithm program because it allows for overflow to dynamically adapt to the changing boundary conditions.

The Genetic Algorithm

The genetic algorithm first calls the previous population (for the first run it would be the initial population) and then calls the results of population's performance from overflow. Overflow does not calculate the pressure recovery directly, so the genetic algorithm takes ρ^*, $\rho^* v^*$, $\rho^* u^*$, $\rho^* w^*$, $\rho^* e_0^*$, and γ and calculates the pressure recovery. The formulas used to calculate p_{0a} and p_{o2} are:

$$p = (\gamma - 1)\left(\rho^* e_0^{\,*} - \frac{1}{2}\frac{\rho^{*2}(u^{*2} + v^{*2} + w^{*2})}{\rho^*} \right)$$

$$M = \frac{(u^{*2} + v^{*2} + w^{*2})\rho^*}{\gamma p}$$

$$p_t = p\left(1 + \frac{\gamma - 1}{2} M^2\right)^{\frac{\gamma}{\gamma - 1}}$$

With the fitness values of each individual, a random number is generated between zero and one and is compared to the probability of a crossover, *pcross*. If the random number is less than the *pcross*, then crossover occurs; otherwise, it skips crossover. If it occurs, then the wheel parent selection function chooses parents for crossover. In the roulette program, a random number between 0 and *sumfitness* is chosen and is compared to the fitness of each individual plus the sum of the previous individuals.[4] When the individuals' sum becomes greater than the random

number, the individual that was last summed is chosen as a parent. Using this technique, individuals with larger fitness values are more likely to be chosen. This is like a roulette wheel, where a die is randomly tossed and has a greater probability of landing in a larger spot. Then a random number between one and seven is chosen to decide where the cut for crossover will occur. The parents swap the parameter after the cut. This process is repeated for each parameter type, location, diameter, and flow rate.

Mutation works in a similar fashion; however, before it occurs, the best fit individual in a generation is spared mutation and crossover and is automatically inserted into the next generation. This technique is called elitism. For the rest of the individuals, if a random number is less than *pmutation*, then another random number is generated to decide where the cut is for mutation. The parameter is mutated after that cut. Mutation randomly reassigns a new value using the same formula for setting a parameter in the initial population. These steps are repeated so that each parameter undergoes this process.[4]

When the new population is created, after crossover and mutation are complete, the individuals are checked to ensure that they satisfy the double location and overlap constraints, in the same way that the initial population generator did.

This new generation is sent to overflow to calculate the behavior. The genetic algorithm runs again and the loop repeats itself until the maximum number of generations is complete.

Results

To date, a sample fitness function has been used to test the genetic algorithm. The fitness function is:

$$y = \left|\cos((x-500)/25) * e^{-|(x-500)/400|}\right|$$

and the graph is:

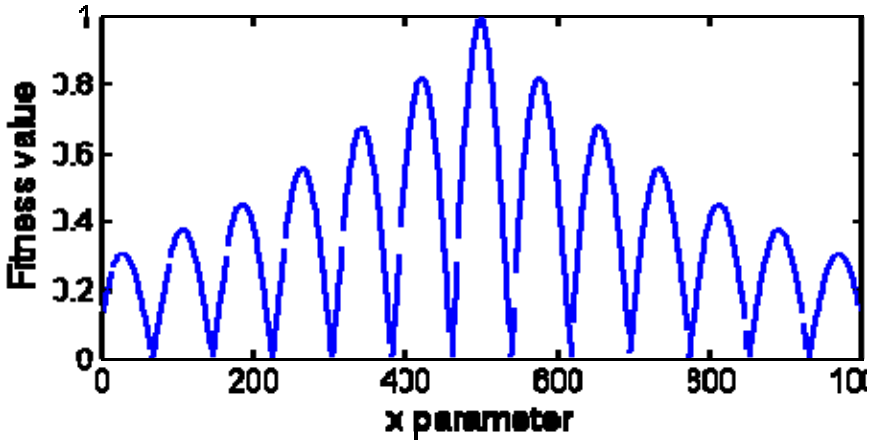

Figure 6. Sample Fitness Function.

The single x parameter was obtained by multiplying a hole location with its diameter and adding the mass flow rate and then summing these values amongst the seven holes in an individual. The optimization was run for eight generations and eight individuals. Higher fitness values are observed as the generations progress. By the eighth generation, the best individual, whose combination of parameters yields an excellent fitness, is repeated, showing signs of converging. The fitness results of each individual of this test are shown below in Table 2.

Testing of bleeding configurations for the hypersonic inlet has not yet been done. A few minor adjustments in the genetic algorithm should enable the overflow simulation to work.

Table 2. Fitness Values for Test Generations.

Generations run from left to right, individuals from top to bottom

1	2	3	4	5	6	7	8

0.113731422	0.234120563	0.117544442	0.235378966	0.07430766520	0.159624562	0.116011731	0.553079069
0.12300083	0.188782021	0.32152468	0.313081115	0.116722323	0.819668055	0.325364143	0.109378308
0.26542297	0.094354488	0.248774186	0.317447215	0.366303593	0.207042605	0.002617246	***0.990552008***
0.195823774	0.031235218	0.07629527150	***0.99905014***	0.236481443	0.26689741	0.250130475	0.250130475
0.179008707	0.578935921	0.158093035	0.607488692	0.387234896	0.456582844	0.799327552	0.339859664
0.115196526	0.554084063	0.026136970	10.06344798	95***0.990552008***	0.575627089	0.129078045	0.454327971
0.052686713	0.039698883	0.056231137	40.306937933	0.250802249	0.489245415	0.555937231	0.492255181
0.214255869	0.26542297	0.578935921	0.578935921	0.382330447	***0.990552008***	***0.990552008***	***0.990552008***

Conclusions

The results from the fitness function show that the genetic algorithm and all its associated programs work. Although this initial run may not be very efficient, it shows promise and good potential for future work. A good next step is to run the sample code with a larger population and more generations, as well as tweaking the code until the results consistently converge faster and accurately. Elitism could be modified to save the two best individuals, creating faster convergence.

When the actual inlet optimization is run, the baseline design, that is the inlet without any bleeding, should be run through overflow and its pressure recovery should be calculated. This gives a standard pressure recovery upon which the program strives to improve. Furthermore, the optimization search can be made more thorough by adding more holes. This would enable more combinations of bleed sections, making the simulation more accurate and more realistic to the actual bleeding configurations carried out in the wind tunnel. The inlet in the wind tunnel has hundreds of small holes. Slots are similar in effect; however, there is not as much flexibility with slots as there is with holes. There is more fine-tuning with holes since you can close or open one hole at a time. However, by adding too many holes, the number of individuals in a population and the number of generations would need to increase significantly, which could have drastic affects on processing time. Considering genetic algorithms have never been applied to the bleed design of a hypersonic inlet, seven holes will suffice.

Three long term applications for this algorithm include adapting the programs for three-dimensional analysis, changing the grid generator so that it dynamically adapts to the size of each hole with consideration of edge effects, and adapting the genetic algorithm for the 40/60 inlet. A partial program has already been written for dynamically changing the grid. It needs more constraints and a few changes in order to fix the problems described in the grid generator section. The 40/60 inlet would be a very interesting problem for the genetic algorithm optimization because it is axisymmetric and is equivalent to a two-dimensional design. It would be a great way to validate the genetic algorithm.

Overall, the using of genetic algorithms with computational fluid dynamics is extremely beneficial. Thousands of cases can be tested while saving the cost and time it would take with a wind tunnel or with other numerical optimization methods.

Appendix

A. How the Batch File Works

Key:
Scripts and pieces of code
Folders, directories
files

1. The **batch script** calls **makegrid,** an executable in the main **inlet** directory. Makegrid calls **bleedgrid**, which creates a single **inlet.fmt** file in the **inlet** directory (the "home" directory. It also creates **ivalues.dat**, a list of i-lines and their x values. **ivalues.dat** is kept in the main **inlet** folder.
2. The **batch script** copies **ivalues.dat** to the **summary** directory in Lou.
3. The **batch script** calls makegrid, which changes **inlet.fmt** to grid.in.
4. The **batch script** moves **grid.in** from the main **inlet** folder to every individual's **inlet#** folder. (Each individual being run through **Overflow** has its own **inlet**# folder.)
5. **The batch script** calls **InitialPop,** an initial population generator in the home **inlet** directory, so that a random population isn't generated every time the **GA** is called, and the rest of the **GA** doesn't run before the **batch script** gets a chance to do anything else.
6. **InitialPop** generates the first population, checks it, and outputs a **gen.dat** file so the **GA** can recover it. It also reads **ivalues.dat**, then assigns i-values to the start and end of each hole in the form of a whole new input file. It outputs each individual's input file as **fort.1,fort.2,…** This file has a .F. for "run from q.restart." since it's the first time.
7. The **batch script** *copies* gen.dat to the summary directory in Lou as **gen**(gen#)**.dat**.
8. The **batch script** *moves* the i-line information files **fort.1,fort.2,…** to the respective **inlet#** folder and calls it **over.namelist**. It also copies each individual's input file into the respective **inlet#** directory in Lou, calling it **inlet**(Generation #).**inp**.
9. The **batch script** calls **overflow** in each individual's **inlet#** folder and outputs **q.save** in the individual's **run#** folder.
10. **q.save** is copied as **q.restart** in each individual's directory.
11. **q.save** is moved to its respective individual's folder in Lou and named **q.save.**(gen#).
12. Each **q.save** is moved to **Inlet** (home directory) and named **fort.**(ind#).
13. The **batch script** runs **GeneticBleed**, the Genetic Algorithm, which reads **gen.dat** (chromosome information), setting values for the oldpop array.
14. **GeneticBleed** finds the fitness values for each individual using the result **fort.**(ind#) files.
15. **GeneticBleed** *creates* **best.dat** file, the chromosome and fitness for the best individual in that population. Each time it is written, there is a statement displaying the number of that best individual and the generation, so that the corresponding q (solution) and grid files can be found.
16. **GeneticBleed** *creates* **fit.dat** file, listing each chromosome number and its fitness.
17. **GeneticBleed** *creates* a new population via crossover and mutation and checks constraints, all within the same program.
18. **GeneticBleed** *outputs/replaces* **gen.dat** for its next run. It also outputs new **fort.1,fort.2,fort.3**… i-line files. These have .T. for "run from q.restart."
19. The **batch script** *moves* **best.dat** into the **summary** directory in Lou, as **best**(gen#-1)**.dat** and does the same for **fit.dat**.
20. The **batch script** repeats itself at step number 7, for gen loops.

Saved in Lou:
ivalues.dat – summary
Inlet(gen#).inp – inlet(ind#)
Gen(gen#).dat – summary
q.save.(gen#) – inlet(ind#)
best(gen#-1).dat – summary
fit(gen#-1).dat – summary

Acknowledgments

I thank my mentor, Dr. Meng-Sing Liou for taking me under his wing and teaching me CFD and its associated programs and May-Fun Liou for her time and devotion to me in helping with Columbia and the batch script. I am grateful to Dave Saunders for providing wind tunnel tours and the hypersonic inlet test data, as well as MaryJo Long-Davis for initiating the idea of a hypersonic inlet project in the first place. I also owe a huge thanks to my colleagues in the 2007 NASA Glenn Academy, especially Ashley Micks. Finally, I thank the Academy staff, Michael Lamberty, Kamara Brown, and program director Dr. Kankam. Their commitment and perseverance to the program, and their personal support cannot go without appreciation.

References

[1] Hill, Philip G. and Peterson, Carl R., *Mechanics and Thermodynamics of Propulsion,* 2nd ed., Addison-Wesley Publicshing, 1992, pp. 217 – 241.

[2] Garcia, A.L., "Programs for *Numerical Methods for Physics* (2nd edition)," URL: http://www.algarcia.org/nummeth/Programs2E.html [cited 6 July 2007].

[3] http://infohost.nmt.edu/tcc/help/lang/fortran/scaling.html

[4] Goldberg, David E., Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley Publishing, 1989, Chaps. 1, 2, 3.