

NOT FOR DISTRIBUTION

OPTIMIZATION OF A MESOSCOPIC PERISTALTIC COMPRESSOR CHANNEL USING A GENETIC ALGORITHM

James Solberg

Marc Sulfridge

University of Illinois at Urbana-Champaign
Department of Mechanical and Industrial Engineering
1206 W. Green St.
Urbana, IL 61801

Abstract

A new type of meso-scale (miniature) compressor, called a peristaltic compressor is described. The advantages of a compressor at this scale are outlined and the importance of the precise shape of the compressor channel is discussed. A method of optimizing the shape of this channel using a genetic algorithm is presented. The adaptation of the algorithm to suit the problem is described in detail, focusing on the many difficulties encountered along the way. The results of the algorithm are then summarized, along with possibilities for future extensions of the work presented.

Introduction

This paper focuses on the design of a mesoscopic peristaltic compressor channel. Thus, in order to understand the material presented, it is necessary to know what, exactly, a mesoscopic peristaltic compressor is. First, the term mesoscopic is used to describe a device that is smaller than a traditional (macroscopic) device, but larger than the recently developed MEMS (micro electro-mechanical systems) microscopic mechanical devices. This scale of compressor is essential for the development of air conditioning systems small enough to be easily portable and efficient enough to require relatively low power to operate, while still being large enough to do appreciable work. The need for such a cooling system is demonstrated in (Philpott, 1998).

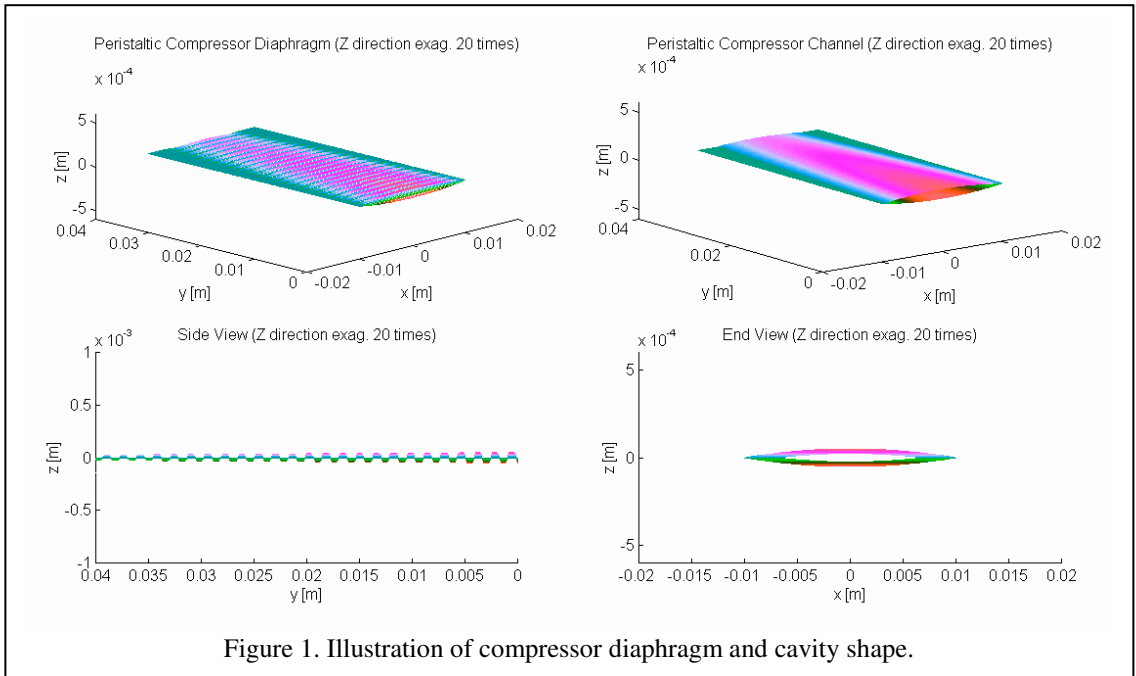
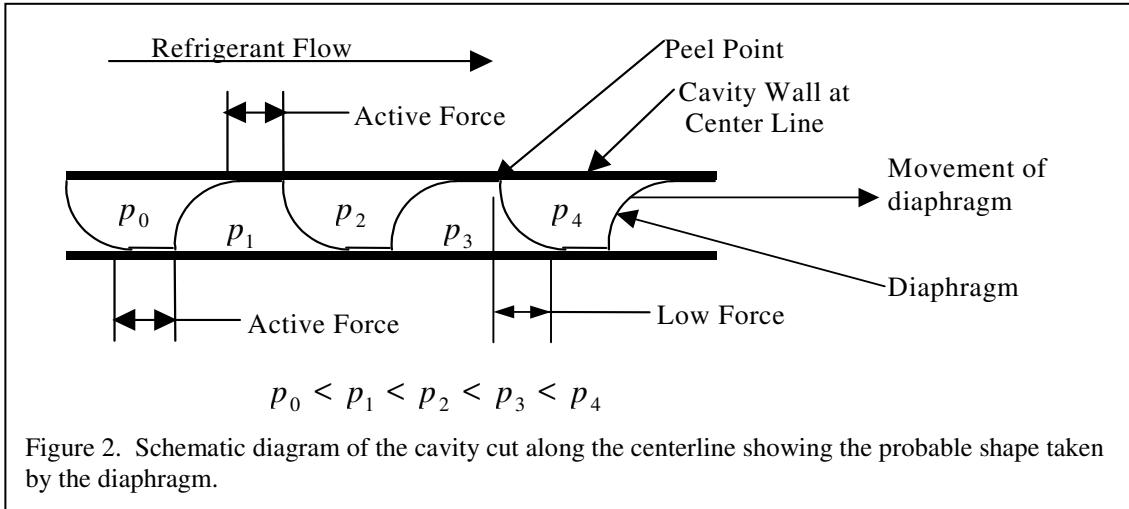


Figure 1. Illustration of compressor diaphragm and cavity shape.

The design under investigation is based on peristaltic action, in which controlled waves are used in rapid succession to propel a fluid down a continuously tapering channel. The idea is to trap a number of pockets of gas using a diaphragm that runs down the center of the channel. Electrostatic actuation is used to force the diaphragm into a wave shape by applying a voltage to a series of electrodes embedded in the diaphragm, and then to make the

wave progress down the length of the channel by modulating the voltage signal applied to each successive electrode. Figure 1 illustrates a basic channel shape and an accompanying diaphragm shape, while Figure 2 illustrates a cross section of a portion of the diaphragm in the channel. Since the channel tapers continuously, the volume of each pocket is reduced as it goes from inlet to outlet. Thus, the pressure in each pocket is smoothly increased as it travels down the length of the channel.



This idea is simple enough, but upon analysis, the true complexities of the design details reveal themselves. The analysis of the channel is based on energy conservation. The proper management of energy is of supreme importance to the design of this compressor. The energy available through the type of electrostatic interaction being used by the compressor is quite small. Thus, the design is largely dictated by energy requirements. Clearly, the only energy input into the compressor is electrostatic. In addition, only half of the energy required to charge up the electrodes is available to do work. The other half becomes capacitive potential energy stored by the closed electrodes.

There are three main sources of energy consumption. These are diaphragm strain, compressive work, and internal viscous friction losses. It is important to note that of these energy sinks, diaphragm strain is different from the other two. It is only required to be input once, not on a continuous basis as are the other three. The reason for this is that once the diaphragm is in its strained configuration, the total amount of strain in the diaphragm no longer changes. Thus, the strain may be maintained by the capacitive potential energy stored in the closed electrodes. The compressive work term is straight forward, as is the internal friction. This friction term turns out to be quite small, and as such it is ignored in the analysis presented here.

The analysis of the two remaining important energy sinks, strain and compression, is simplified somewhat by the fact that they can, to some extent, be considered separately. In fact, the strain energy analysis has been conducted analytically. This is very useful, because it places some rather strict bounds on how large the maximum dimensions of the compressor channel can be, and also bounds the number of wave pockets that can be used, since more pockets means more strain. In addition, manufacturing issues place lower bounds on these dimensions. The strain analysis also provides the functional form of the cross sectional shape of the diaphragm, since it clearly shows that the minimum energy shape of a diaphragm under uniform distributed load (pressure gradient in one direction, and capacitive force in the other) is parabolic. All of this analysis is described in great detail in the Master's thesis of Marc Sulfridge (Sulfridge, 1998). However, this thesis is currently being withheld pending patent review, and most likely will not be released until the spring of 1999. The results of that analysis will be used in this document without further discussion. Thus, we are left with only two issues undecided in designing the compressor channel. How, exactly, should the height and width of the diaphragm channel taper from inlet to outlet, and in how many stages (i.e. how many pockets) should the compression take place? To answer these questions, one must turn to compressive energy analysis.

It turns out that this analysis is not straightforward for a number of reasons. The most important issue is that the behavior of a gas suitable for use in a refrigeration loop is rather erratic. This is due to the fact that it must be used in and around its two-phase region. When a fluid goes from saturated vapor to two-phase vapor, there are sharp discontinuities in many of its defining properties (such as temperature). Thus, it is impossible to define all of its properties with a single functional form. This makes any attempt to use gradient methods very difficult. Further

complicating the issue is that fact that one of the parameters being optimized, namely the number of pockets to be used, is an integer variable. Anyone who has attempted to achieve a precise gear ratio using gears with (obviously) integer numbers of teeth, will understand the difficulty that this presents. Other issues, such as the effects of heat transfer on a compression process at this scale are too complicated to go into in detail (see (Sulfridge, 1998) for a presentation of the effects of heat transfer during compression), but they also conspire to make the problem more difficult.

Genetic algorithms offer a solution to handling many of these complexities. Genetic algorithms do not rely on gradients in order to optimize. A genetic algorithm can also handle an integer variable just as easily as it handles continuous variables. Clearly, there is a strong motivation to attempt to use a genetic algorithm to optimize the compressor channel design.

This paper will present the development of just such an algorithm. Emphasis will be placed on the difficulties encountered along the way and the methods by which these difficulties were overcome. The evolution of the algorithm will be described in detail, including the development of operators and of the objective function. Results of several versions of the algorithm will be presented in order to demonstrate why and how the algorithm was adapted. It will also be shown how important proper understanding of the true building blocks was to the development of a good algorithm. Finally, the results of the genetic algorithm will be summarized, ramifications will be considered, and possible extensions of the algorithm will be discussed.

A starting point

The initial algorithm used for this project was based heavily on (Cao, 1997). The reason why this algorithm was originally chosen is that it offered a method of handling both continuous and discrete (i.e. integer) variables in the same algorithm. To be precise, then, the initial algorithm used was really a form of evolutionary programming. While the details of the operators used will be discussed later, one thing that is important to keep in mind is that this original algorithm did not incorporate crossover of any kind. Instead, it used a directed form of mutation in which the magnitude of the mutation was determined by the relative fitness of the individual being mutated. While a simple crossover operator was eventually added to the algorithm, this original mutation operator was used in every incarnation of the GA.

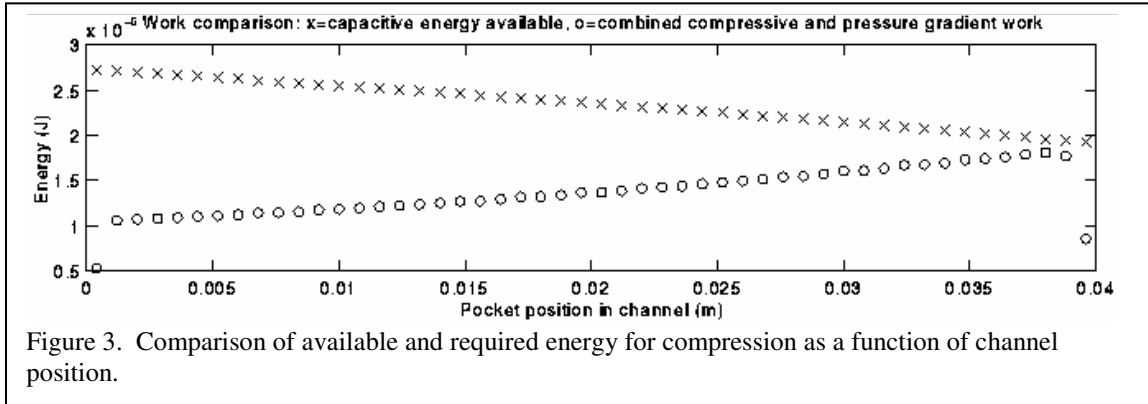
The encoding of the problem was originally going to be a string of real numbers representing the coefficients of either a polynomial or a Fourier series which represented the functional shape of the height of the channel from inlet to outlet, followed by a similar encoding for the width of the channel, and finally an integer representing the number of channel pockets. However, this encoding proved problematic in that the coefficients of the various terms would be of widely disparate order, and thus it would be difficult to randomly initialize the original generation in such a way that the entire reasonable search space was covered with certainty. However, as mentioned above, the strain energy analysis coupled with manufacturing issues provided a nice set of bounds on the channel height and width. Thus, it was decided that the aforementioned coefficients should be replaced by actual heights and widths at a discrete number of evenly spaced points down the length of the channel. The number of points was arbitrarily chosen to be one hundred, initially. This was later determined to be far too precise, and as such it was eventually reduced to twenty.

The next step was to devise a means of providing the initial generation to the algorithm. This was naively carried out by randomly selecting widths and heights uniformly between the minimum and maximum allowable values. This proved to be a rather severe mistake caused by a fundamental lack of understanding about what the true building blocks of a good compressor channel were. This will be discussed in more detail in the next section.

After creating an objective function, which will be discussed in the next section, along with extensive debugging of the code, the GA was given a test run. This early test run provided two crucial pieces of information. The first, not surprisingly, was that even a poorly chosen genetic algorithm will rapidly improve a given fitness function. Unfortunately, the second, more important lesson was that just because the fitness was improving, that did not mean that the 'best' individuals were actually good, it just meant that they had a better fitness.

The objective function

Before the analysis presented in this paper was done, the compressor channel taper shape was just assumed to be linear in height and width, and the number of pockets was chosen arbitrarily. The problem with these assumptions is illustrated quite nicely in Figure 3. This plot depicts available electrical energy at each pocket on one curve, and required compressive work on the other. Notice how the two curves converge, so that the available energy curve must have a huge margin (provided by applying excessive voltage) at the entrance in order to have any margin at all at the exit. Clearly, this is far from optimal.



Choosing an appropriate objective function proved to be quite a daunting task. The first step was to decide what needed to be optimized. In order to quantify how, exactly, the channel should taper, it was necessary to go back to energy balance. The key thing to notice with this compressor design is that the electrostatic energy available at any given point in the channel, and the pressure work required at that point in the channel are both dependant on the channel geometry at that point. The electrostatic energy is proportional to the surface area of the channel, while the pressure work is proportional to some function of the volume and pressure of the channel. For energy balance, what we would really like is for the available energy to vary in exactly the same way as the required work (plus some safety factor). Thus, an important method of quantifying the fitness of the channel would be to make it proportional to the standard deviation between the available energy minus its average value and the required work minus its average value. The reason for subtracting out the average values is so that we are just considering how the two curves change, and not penalizing a given channel for having a large safety margin.

In addition to the fact that it is desirable for the two energy terms to change in a similar fashion, it is also advantageous to have the required energy be as small as possible. Thus, an additional term was added to the fitness function representing the sum of all required compressive work terms through the channel. However, this condition is subject to the constraint that a compressor should be able to achieve a desired pressure. The farther from the desired pressure value the compressor actually achieves, the worse its fitness should be. Thus another term was added to the objective function representing the distance of the actual output pressure from the desired output pressure.

One further constraint was that the available energy should never be less than the required energy. If this constraint were violated, then the compressor would simply cease to function at the point where the violation occurred. Thus an additional term was added to the objective function which took this into account. It should be noted that all of these terms are added together in the objective function in a similar manner to the way one would add constraint equations to a function to be optimized using Lagrangian multipliers. This sort of penalty method for handling constraints has been discussed in great detail in (Smith, 1993). Thus, each constraint must be scaled by a constant so that no one term always dominates the value of the objective function. These coefficients were determined through trial and error.

This section has outlined the objective function as it was used in the final incarnation of the GA except that it is missing one term. The need for this last term became apparent after the naively executed first run, and it is discussed in the next section.

Building blocks

It was not until a fundamental understanding was gained of what the important schemata were in this real coded GA that true progress was made on getting good results from the genetic algorithm. Initially, building blocks were not even considered in designing the algorithm. It was just assumed that the problem would be given to the GA, and after a few hours of computation, a good result would come out. Unfortunately, this did not take place. Instead, after a few hours of computation, a lousy result came out. Almost immediately, the question of fundamental importance was asked: What went wrong?

While there was no single answer to this question, one of the key parts of the answer lay in understanding the building blocks. As it turns out, the original method by which the algorithm was initialized did not provide a sufficient number of these important schemata. In the encoding used, the most important schemata are of defining length two. They represent a distance from the centerline of the channel (for the width this would be in the

horizontal direction, and for the height it is vertical) and a slope. By piecing together these simple schemata, any basic channel shape can be obtained.

But in the case of a compressor channel, one obvious piece of information was completely ignored in the encoding. That bit of information is that the compressor volume should decrease monotonically. This is because any increase in volume would simply cause the compressed gas to re-expand, thus wasting all of the energy that was required to compress it. It was decided that this problem would be handled in the method by which the first generation was initialized. Instead of randomly choosing every point down the length of the channel, a random initial height (or width) scaled to the maximum and minimum allowable values is selected, and a random slope (again appropriately scaled) is also selected, and these are used to construct a line. With a sufficiently large population size, a wide range of the above described schemata will be created using this initialization, and all of the resulting individuals will be monotonically decreasing. However, with crossover and mutation, it is possible that some portions of a given individual could translate into volume increases. To combat this, an additional term was added to the objective function, equal to the absolute value of the sum of all negative pressure work terms. This helps to ensure that any channel shape that causes re-expansion will have a poor fitness.

Mutation

The initial iteration of the solution for our problem adopted a form of mutation that was the only means of parameter space exploration (Cao, 1997). This mutation plays a very significant role on the real-coded alleles. This form of mutation perturbs every allele by some relatively small amount about its original value. A parent vector x_I is mutated to create an offspring vector x_{I+N} . A mutation is done by adding to each component of $x_{I,J}$ a Gaussian variable with zero mean and a standard deviation proportional to the scaled objective values of the parent trial solution, that is:

$$x_{I+N,J} = x_{I,J} + N(0, \sigma_J^2), J = 1, 2, 3, \dots, v \quad \text{where} \quad \sigma_J^2 = \beta_I \frac{f_I}{f_{\min}} \times (x_J^u - x_J^l)$$

Where $x_{I,J}$ denotes the J th element (J goes from 1 to v ; where v is the number of variables for each individual) of the I th individual. $N(\mu, \sigma^2)$ represents a Gaussian random variable with a mean μ and variance σ^2 . f_{\min} is the minimum fitness in the population. β_I is a constant of proportionality to scale f_I/f_{\min} . Every variable is mutated according to its relative fitness. So, on average an individual with a relatively small fitness will be mutated less from its original value than an individual with a larger fitness value (The fitness is being minimized.).

Because of this drastic mutation of every individual, it is much more likely that the new (mutated) individual would have a worse fitness value. For this reason a $(\mu+\lambda)$ -strategy was used to ensure that each iteration of the algorithm took a step in the right direction (each generation does no worse than its previous). With a $(\mu+\lambda)$ -strategy μ parents create λ descendants by recombination and mutation and the μ best individuals out of parents plus descendants are deterministically selected to become parents of the new generation (Back, 1995). For this mutation operator μ parents create μ (mutated) individuals.

Crossover

Because of the initial blind faith that was naively placed on the power of our initial evolutionary program, crossover was not initially included as an operator. A similar form of the initial GA that used only mutation and selection had been used to optimize a mechanical system that seemed to be similar to our problem. This GA was then expected to explore and find a global optimum by using only the mutation described earlier and $(\mu+\lambda)$ -selection (where $\mu=\lambda$, and every individual was mutated exactly once). But when we tried to adopt this algorithm to our problem, mutation alone did not yield results that were close to what was expected.

The original algorithm for our GA initialized the first generation in a totally random fashion. Every height, width, and number of channels was chosen randomly between their respective minimum and maximum heights and widths. This method, as has been previously noted, left little chance for any appreciable amount of building blocks to be present in the initial generation. Since the only mode of improvement was mutation, not having numerous and diverse building blocks in the initial generation was not an issue. But, once it was realized that mutation (without selection) was not enough, the original initialization scheme needed to be modified to accommodate crossover. As mentioned earlier, it is known that the volume of the channel must never increase in the direction of fluid movement (and still be efficient). This information was used to initialize the first generation. All of the initial heights and widths were set to linearly decrease between two random numbers (both numbers are scaled by the minimum and maximum heights and widths).

For example, let the minimum and maximum for any point in the channel be 1 and 10 respectively. The first thing done is to choose a random number scaled by the range 1 to 10; say 5.67. Then, another random number is chosen between 1 and 5.67; say 2.45. So, the height at the low-pressure end of the compressor channel (where the fluid enters) would be 5.67, and the height at the high-pressure end of the compressor channel would be 2.45. A linear spline would represent the intermediate points. If the channel is partitioned into n sections, the cross-sectional heights in the direction of the fluid flow would be $\{2.45, 2.45+(1*(5.67-2.45)/n), 2.45+(2*(5.67-2.45)/n), \dots, 2.45+((n-2)*(5.67-2.45)/n), 2.45+((n-1)*(5.67-2.45)/n), 5.67\}$. Many more valuable building blocks can be created in the first generation using this initialization technique.

This new initialization paved the way for crossover. The next issue addressed was what type of crossover to use. Initially, multiple point simple crossover was chosen. The number of crossover points would be randomly chosen. It was later concluded that single point crossover was superior to multiple point crossover because the greater the number of crossover points used, the more disruption the building blocks would experience. For example with multiple point crossover, it would be likely that both height and width would be modified, but exceedingly unlikely that both would be modified in a positive fashion.

Selection

The selection scheme must implicitly (or explicitly) surrender to the facts that it is unlikely that crossover will produce a better descendant than its respective parent and it is much less likely that any single mutated descendant will be superior to its parent. As previously mentioned a $(\mu+\lambda)$ -strategy will be employed where μ parents create λ descendants by crossover and mutation and the μ best individuals out of parents plus descendants are deterministically selected to become parents of the new generation. This selection operator will mutate every parent exactly once to yield μ descendants and will perform crossover exactly once to every parent to create μ more descendants. So the selection process must select μ individuals from 3μ (μ original, μ mutated, and μ crossed) if there is to be a constant population size (where μ is the population size).

The first thing selection does is to randomly group all of the individuals into predetermined group sizes. Then the total fitness of the group, f_{group} , is determined by summing all of the fitnesses in the group. Next, every individual is given a weight equal to its fitness divided by the fitness of its respective group. Finally, these new weights directly compete against everyone in the entire population by only allowing the individuals who are in the lowest third to pass to the next generation.

For example with a population of size 4 after crossover and mutation there are now 12 individuals with which 4 need to be chosen from. Let's say the fitness value of each of these individuals corresponds to a number in the vector $\{13 \ 10 \ 24 \ 1 \ 9 \ 5 \ 3 \ 4 \ 8 \ 14 \ 20 \ 2\}$. This group is then broken up into groups; let's say 3 groups of four: group 1 = $\{1 \ 24 \ 10 \ 13\}$, group 2 = $\{2 \ 20 \ 14 \ 8\}$, and group 3 = $\{4 \ 3 \ 5 \ 9\}$. Then f_{group} for each group is 48, 44, and 21 respectively. Next, each individual is given a weight between zero and unity that relates its fitness to the fitness of the other members of its local group. The weights for each group were then found to be: group 1: $\{1 \ 24 \ 10 \ 13\}/48 = \{0.0208 \ 0.5000 \ 0.2083 \ 0.2708\}$, group 2: $\{2 \ 20 \ 14 \ 8\}/44 = \{0.0455 \ 0.4545 \ 0.3182 \ 0.1818\}$, and group 3: $\{4 \ 3 \ 5 \ 9\}/21 = \{0.1905 \ 0.1429 \ 0.2381 \ 0.4286\}$. After every individual is given its weight, the individuals that have a weight in the lowest third of all of the individuals is passed on. In this example the individuals that correspond to having a fitness in the vector $\{0.0208 \ 0.0455 \ 0.1429 \ 0.1818\}$ will be passed to the next generation. Table 1 summarizes the results.

group	fitness - rank	weight - rank	selected
1	1 - 1	0.021 - 1	YES
1	24 - 12	0.500 - 12	no
1	10 - 8	0.208 - 6	no
1	13 - 9	0.271 - 8	no
2	2 - 2	0.046 - 2	YES
2	20 - 11	0.455 - 11	no
2	14 - 10	0.318 - 9	no
2	8 - 6	0.182 - 4	YES
3	4 - 4	0.191 - 5	no
3	3 - 3	0.143 - 3	YES
3	5 - 5	0.238 - 7	no
3	9 - 7	0.429 - 10	no

Table 1: Numerical example of selection

While the individuals with the three best fitnesses did get selected, the individuals with the 4th and 5th best fitnesses did not make it into the next generation while the individual with the 6th best fitness did make it. One can imagine a scenario that even the very best individual does not make it into the next generation. With a small group size and if all of the individuals in the best individual's group have similar fitnesses, then there is a finite probability that the best individual will not make it to the next generation.

Because this selection scheme is not a traditional method of selection, parameters such as takeover time and selection pressure were not analytically assessed. This specialized selection method does seem to have traits from both tournament selection and ranking (Goldberg, 1991). But duplicate copies are never distributed to the next generation (unless by pure coincidence). Our algorithm can afford not to duplicate the better individual because the best individual will almost always make it to the next generation no matter how crossover and mutation disrupt it.

One way to vary the selection pressure is to change the size of the groups that selection uses to create the relative weights of the individual. In order to decrease selection pressure decrease the group size. This will give individuals with worse fitnesses a better chance of propagation. This is beneficial because these individuals with a worse fitness may contain valuable building blocks that can later be recombined into something good (Goldberg, 1989). These building blocks would have otherwise been eliminated by selection. But decreasing the group size also increases the amount of noise that selection causes. Recall the example of selection described earlier. If the group size was decreased to just 2, and if the individual of fitness 1 was paired with the individual of fitness 2, then the individual of fitness 2 would have no chance to propagate to the next generation. If it were the case where the two best individuals were paired together and had very close fitnesses, then there is a good chance that neither one would survive!

Because this algorithm uses a multi-objective fitness function (we are trying to minimize multiple parameters simultaneously), the selection pressure contribution of each parameter can be varied by multiplying it by a factor or raising it to some power. The greater a specific parameter of the total fitness function is, the more selection pressure that parameter imposes. As discussed in detail earlier, if a fitness function attempts to minimize five different and equally important parameters then it is desirable to maintain each of the five parameters at relatively equal magnitudes (i.e. keep all five equal to each other).

Sizing

Because of the nature of real-coded genetic-algorithms (GA), an accurate prediction of the population size is complex. This is because a real-coded GA such as this one has a parameter space that is essentially the number of variables per individual times infinity (in this case infinity is actually the number of unique floating point numbers between the minimum and maximum value of each variable which is a function of your floating-point precision). This results in an infinite number of schemata (Goldberg, 1991a). Once it was established that crossover is the dominant form of creating a better individual, a diverse and numerous pool of building blocks must to be present in the initial generation in order to attempt to explore as much as the parameter space as possible. Mutation is able to explore new building blocks that are not included in the initial generation or obtainable by crossover alone.

The population size was of course constrained by hardware capabilities. Population size was determined by the largest possible population size that could converge to a minimum fitness in a number of generations that could be computed in a practical amount of time. This was determined experimentally through an iterative process. Clearly, the above mentioned methods of modifying selection pressure also have an influence on the time to convergence.

The Algorithm

Once all the operators and other subfunctions have been implemented, the main function should fall into place:

```
define popsize, maxgen, maxheight, maxwidth, minheight, minwidth, no_of_pockets
// original is a matrix where each row is an individual of that generation.
original = initialize(popsize, maxh, minh, maxw, minw) // initial generation
new_fit = fitness(original) // vector containing fitnesses of original
while g<= maxgen
  org_fit = new_fit;
  variance_mat = variance(org_fit) // calculate the variances of original for mutation
  mutated = mutate(original, variance_mat); // mutate original
  mut_fit = fitness(mutated); // calculate the fitness of each individual of mutated
  crossed = xover(original); // perform crossover on original
  xvr_fit = fitness(crossed); // calculate the fitness of each individual of crossed
  [new, new_fit] = select(original, mutated, crossed, org_fit, mut_fit,
                        xvr_fit); // ( $\mu+\lambda$ )-selection; new is the new population after selection
```

```

original = new; // set original population to the new population for the next generation
if min(new_fit)<fmin
    then fmin = min(new_fit) // keep track of the best individual
end
g=g+1;
end

```

So, a single generation can be represented as:

```

original -> mutated
original -> crossed
{original, mutated, crossed} -> new
original = new (start new generation)

```

The nuts and bolts

After the algorithm was developed, it had to be implemented in some fashion on a computer. Since two Pentium II, 350 MHz computers with 64 MB of RAM were readily available, it was decided that they would be ideal platforms for executing the algorithm. It was then necessary to choose a programming language. MATLAB was eventually chosen as the programming environment for a number of reasons. The most important reason for using MATLAB was its superior ability to manipulate vectors and arrays. This ability is indispensable when dealing with thousands of GA strings. Additionally, its programming structure is syntactically very relaxed, making algorithm implementation relatively simple. Also, many useful functions are built in to the MATLAB environment, saving time when putting together the GA. Finally, MATLAB has a number of native plotting abilities, making visualization of the results particularly simple.

Results

While the development of the genetic algorithm is quite challenging in its own right, if it does not produce functional results, then it is really quite useless. Unfortunately, as has been alluded to many times already, the early results achieved by the GA in its original form were absolutely terrible. A sample of the results from this initial algorithm is shown in Figure 4. Note the horribly jagged structure of the channel. None of the objectives were even close to achieved. However, the extremely poor results could not possibly be anticipated by looking at the plot of minimum fitness provided in the same figure. It behaves exactly as is expected by a good GA, in that it exponentially approaches an optimal value. It is just that the optimal value is not very good. The nature of this failure has been explained rather thoroughly earlier in this paper.

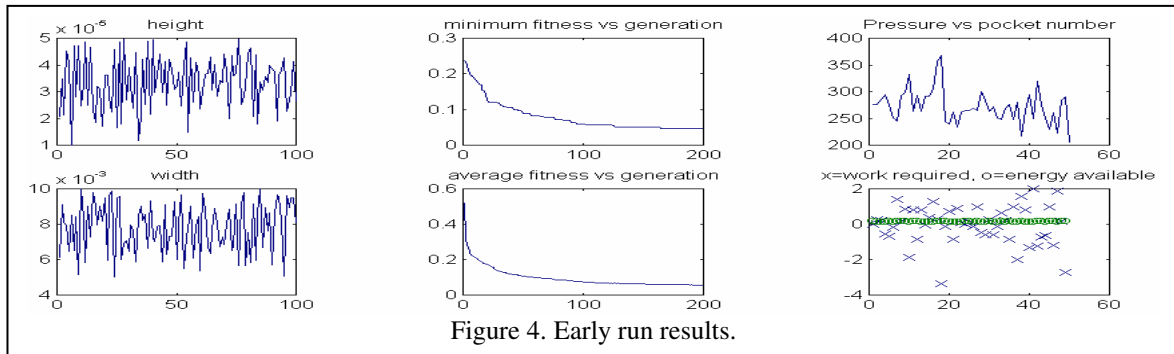


Figure 4. Early run results.

Moving on to a later, intermediate result, in which the algorithm being used is basically the same as in its final incarnation (with the exception that the coefficients on the terms in the objective function have not yet been refined) several things stand out. These results are depicted in Figure 5. First, it can be seen that the minimum fitness once again behaves quite nicely. Note that the absolute value of the fitness function depends on the scaling of all of the terms comprising it, so that only the relative fitness improvement should be compared for the three runs shown. Also, due to the new implementation of the initialization function, the heights and widths obtained are far less chaotic. However, the variance between the energy available and the energy needed is still quite large, although in this case the desired pressure has more or less been achieved. These are clear indications that the algorithm is now on the right track, but that further refinement of the various coefficients on all of the fitness terms is necessary. Some of the terms are scaled in such a way that they are almost insignificant early on when the widest diversity of schemata are present, but later on, when the only real mechanism for obtaining new schemata is through mutation these terms start to have more emphasis. This problem cannot be solved simply by multiplying by a constant.

Rather, in order to adjust the relative scale of these terms at various times in the run, they can be raised to some power and multiplied by a constant.

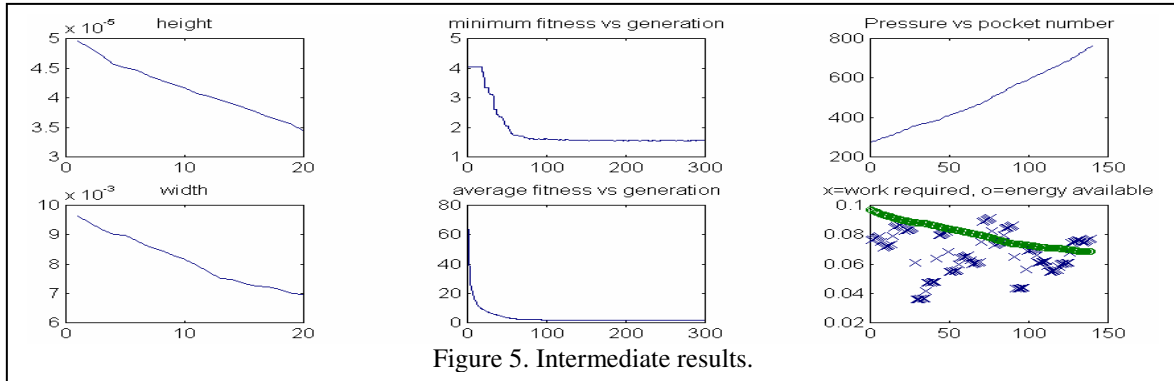


Figure 5. Intermediate results.

The results of this experimentation can be seen in Figure 6. Once again, the fitness improves as expected, but this time the other terms are far more refined as well. First, desired outlet pressure is almost perfectly achieved. This is very important, as the device is a compressor. Second, and just as important, the required energy never goes above the available energy. This is an essential result, because if it did not hold true, then the compressor would simply not function past the point in the channel where the constraint was violated. The diaphragm would peel away from the channel wall, and that would be as far as it would go. In addition, and this is where the GA has truly accomplished something, the variance between available and required energy has become quite small. This allows the use of a smaller operating voltage (and thus a higher operating efficiency) to achieve compression, and allows for a nearly uniform energy margin at every point in the channel. While this result is obviously not perfect, it is better than any prior channel shape considered before this analysis was performed.

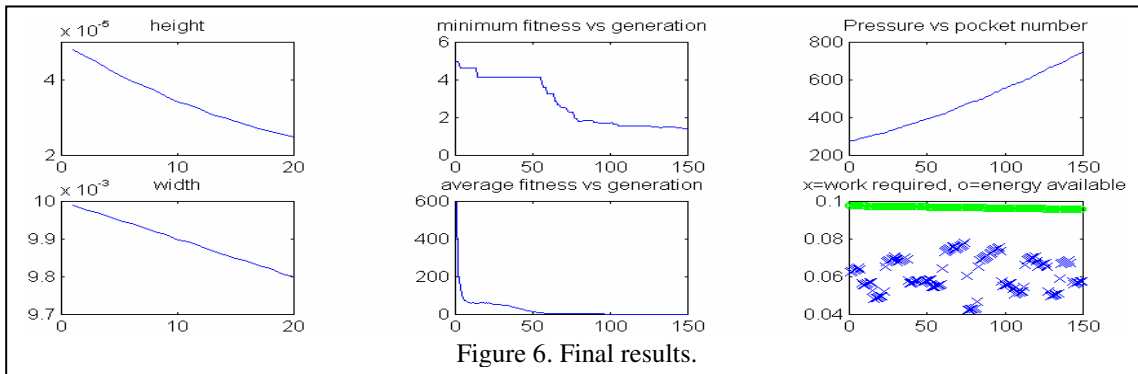


Figure 6. Final results.

Ramifications

The impact of the results of this analysis on the implementation of the mesoscopic peristaltic compressor cannot be overstated. In fact, a slightly refined version of the best solution obtained in this project is now being considered for instituting the first prototype of this compressor design. The refinement was simply to smooth out some of the bumpiness in the channel by doing a low order polynomial fit to the channel height and width values that came out of the GA. By doing so, the uniformity of the two energy curves was made almost perfect. While the prototype has yet to be built, let alone empirically tested, this project has provided a channel design that has the best chance of success yet considered. In other words, this work has supplied a channel design superior to any previously obtained.

Extensions

As discussed earlier, since there is a finite population size, only a finite number of building blocks can be explored. But with this real-coded GA there is theoretically an infinite number of building blocks. So the absolute global optimum can never be exactly known using this GA (assuming infinite precision). This may be due to the solution resolution (which is a function of all the schemata obtainable by a single run of the algorithm and the total number of schemata that can theoretically be formed). Even with a large population size there may still be a “needle in the haystack” that we are missing. To make things worse many of the best building blocks may be selected out of the population because the rest of the schemata of that particular individual caused the fitness to be so bad that it is not selected. So how does this GA get any good results? By making the population as large as possible. Also,

mutation does do a large amount of parameter searching (much more than say a simple GA) that could never be explored by crossover alone. But it is actually a rare event that mutation finds a good individual.

Using practical processing times, another form of search needs to be implemented. While GA's are great for searching large and diverse parameter spaces, there are more efficient ways of doing hill-climbing once there is a global optimum in sight. So, in order to fine-tune this algorithm, some sort of specialized hill-climbing technique (gradient-based) could work wonders for the solution. Once the GA finds a good individual, and it seems like the GA can no longer improve on it (with its current resources), then the GA could pass this individual to a local hill-climbing subalgorithm (Goldberg, 1989). This GA hybrid is only a proposed extension and is not within the scope of this paper.

Summary

The design of a mesoscopic peristaltic compressor channel exhibits multi-modal, multi-dimensional, and complex optimization. Many of the techniques used in the original iteration for the algorithm were borrowed from an evolutionary programming scheme that was thought to be able to scale to our problem (Cao, 1997). While we knew that evolutionary programming would be the road to eventual success, we did not foresee the problems with the borrowed algorithm.

Mutation was only slightly modified. This operator is unique in the fact that it mutates every allele of every individual. Individuals with better fitnesses are mutated less to attempt to preserve those building blocks. The original crossover was completely thrown out. Simple single point crossover replaced it. Single point crossover then became the dominant creator of better individuals. As a result, the initialization scheme had to be modified. The original initialization created individuals randomly which left little chance for any particular individual to have good building blocks. Knowledge of the problem was imposed in the initialization scheme to force good building blocks to be present in the first generation.

In every generation the original population (either the initial population for the first generation or the selected population of the previous generation) first be would be mutated. Then, the original population would have crossover performed on it. Finally, the mutated, crossed, and original populations would compete for spots in the next generation. The selection operator had characteristics of both tournament and ranking selection. By combining all of these operators, an efficient and effective GA was created which achieved results superior to any that had been obtained prior to this project.

References

- Back, Thomas & Schwefel, Hans-Paul (1995). Evolution Strategies I: Variants and Their Computational Implementation. *Genetic Algorithms in Engineering and Computer Science*, Chapter 6, 111-126. John Wiley and Sons
- Cao, Y. J. & Wu, Q. H. (1997) Mechanical Design Optimization by Mixed-Variable Evolutionary Programming. *1997 IEEE international Conference on Evolutionary Computation (ICEC '97)*, pp 443-446
- Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, MA: Addison-Wesley
- Goldberg, D. E. (1991a). Real-coded Genetic Algorithms, Virtual Alphabets, and Blocking. *Complex Systems*, 5, pp 139-167
- Goldberg, D. E., Deb, Kalyanmoy, & Clark, James H (1992). Genetic Algorithm, Noise, and the Sizing of Populations. *Complex Systems*, 6, pp 333-362
- Goldberg, D. E. & Deb, Kalyanmoy (1991). A Comparative Analysis of Selection Schemes Used in Genetic Algorithms. *Foundations of Genetic Algorithms*.
- Philpott, M., et. al. (1998). Phase I Report: Integrated Mesoscopic Cooler Circuits (IMCCs). University of Illinois.
- Smith, A. E. & Tate, D. M. (1993). Genetic Optimization Using a Penalty Function. *Genetic Algorithms*.
- Sulfridge, M. A. (1998). The Mesoscopic Peristaltic Compressor. University of Illinois M. S. Thesis, Urbana, IL.