Universal Adaptable GA with Engineering Help Tutorial

James G. Schiiller, Greg L. Vitko, and Christian P. Wagner
Department of Industrial and Systems Engineering, Oakland University, Rochester, MI, United States

Abstract - *The Many treatises on genetic algorithms (GA) analyze particular methods of developing problem specific GA and few try to develop a comprehensive analysis of combinations of all methods into a universal framework. Such a universal framework could be used to develop what could be called a Universal Adaptable GA. One of the problems of GA is that no one exactly knows all the details. These can be the ever-changing parts and cannot be made static at this time until they are more fully understood. Other areas are well understood, these should be the static parts in the universal adaptable GA. The goal of this work is to explain both general and problem specific areas by an example with a variant of Tetris as case study to demonstrate how a GA can solve problems. This example will make GA easier to understand and replicate.*

Keywords: Artificial Intelligence in Games, DNA, Genetic Algorithm, Tetris

1 Introduction

The development of faster computers in the twenty-first century has made it possible for computers to explore their environments and create solutions for many types of problems without much human intervention. The computer is no longer seen as a big storage bin, but as a useful supplement to help people solve complex problems, on condition: we have to give the computer the right set of instructions. Coming up with these right set of instructions is a problem. The instructions given have to be both general and specific enough to satisfy the minimum requirements for responding to many different types of problems. Responses do not have to be perfect. The universal adaptable genetic algorithm (GA) has been proposed as a suitable method to apply to get the computer to respond to many different types of problems for us.

Before continuing on to having the computer respond to some challenging problems, we have to have a clear understanding of how a computer GA exactly works, absent from all the common misunderstandings. After understanding, we can proceed to deal with more difficult aspects of the GA and see why developing the universal adaptable GA is a good idea. It is the high hopes of this research to start to assemble a universal adaptable GA, for other researchers to adjust and expand and apply to larger scale problems. How this is done is by demonstrating an application of GA to a variant of the game of Tetris.

2 Quality of Solutions

The computer can be used as a resource to solve real problems by applying a GA and running it to speed up millions of years of evolution by simulation within an hour's amount of time. The result is the creation of an "individual" who can solve problems in a real environment. In order for the computer to be resourceful for us, we have to first understand the GA. Knowing what quality of solution it seeks is a good place to start.

The GA aspires to converge on the equilibrium solution to satisfy the minimum requirements of a problem. This is an adequate compromise when facing new problems. The GA does not normally seek to converge on perfection, so there has to be this compromise of a robust solution but not necessarily the optimum solution. For example, a genetically bred thoroughbred horse can be seen as a genetic solution for short distance speed, but even champion thoroughbreds have been defeated in extreme distance horse races by "mutt" American mustangs without as specialized short race breeding. The point is the realistic goal of any artificial general intelligence method including GA is to find the mutt.

3 Genetic Algorithm

John Holland discovered the GA in terms of AI for applying it to computers [1]–[3]. Holland first started looking around at how the GA even works. We know it takes place from sex. There is the DNA strand from the father and the DNA strand from the mother. Somehow two DNA strands come together and create a child. The first questions he asked then were "What's the mechanism by which this happens? What do you do? How do you get these DNA strands to work? What's the mechanism behind it?"

3.1 Genetic Algorithm Outline

The overall idea of what is going on with genetic learning is shown in the steps below. These are the steps in genetics given to the computer to use genetics to create intelligence.

The steps are as follows:

1) Define what a DNA strand is for our setting AND we have to define an interpreter that transforms any DNA strand into an individual that performs in the environment.

2) Randomly create a population of *N* individuals (e.g.,1,000 individuals).
3) Now we have to determine the fitness of any individual.
4) We create a new population of 1,000 by creating pairs of parents to produce one offspring 1,000 times. Pick two parents at random where the fitness of each parent affects the likelihood that it is selected (the better the fitness, the more likely it is selected). When the two parents are selected, the new child is determined by genetic operators: e.g., crossover, inversion and mutation.
5) Keep the new population of 1,000 and discard the old (many variations exist – e.g., keep the 5% best, can one individual be father and mother?).

The GA has a simple outline, yet the devil is lurking in the details. Keep in mind this is an oversimplification of how real DNA works in biology.

3.2 Mapping the Outline

Step one in the preceding outline asks us to define the DNA strand. The definition of the DNA strand in the universal adaptable GA is shown below. The words DNA strand, chromosome, organism, and individual are interchangeable words. The length and number of organisms are scaled down for illustration. Contrast the number of individuals below with the example number given in the outline. There would normally be more than 10 initial organisms in the initial population, as the steps in the GA specified there should be 1,000 individuals. Additionally, the DNA strands below only have a chromosome length of 10. The size of the organisms listed is a gross simplification of DNA in GA. Remember the goal is to avoid all the common misunderstandings. Some definitions have to be defined and simplifications made on smaller scales, to see the respective computer model part matched to the equivalent part in the biological genetics.

$$\begin{aligned}
\text{organism}_0 &= (0000000000)\\
\text{organism}_1 &= (0101010101)\\
\text{organism}_2 &= (1010101010)\\
\text{organism}_3 &= (1110000111)\\
\text{organism}_4 &= (0011001100)\\
\text{organism}_5 &= (1100110011)\\
\text{organism}_6 &= (1010101010)\\
\text{organism}_7 &= (0011111100)\\
\text{organism}_8 &= (0000000000)\\
\text{organism}_9 &= (0101010101)
\end{aligned}$$

Each gene in the organism is equivalent to a pair of zeros in the first organism$_0$ = (0000000000) <=> (|00|00|00|00|00|), where the '|' delimits the beginning and end of a gene. The full four letter genetic alphabet ACGT can be represented the same way in a computer using the following binary assignments: A = 00, C = 01, G = 10, T = 11.

Everything we perceive can be encoded in "0's" and "1's." Computers are only limited by the hardware, thanks to current software implements of larger integer values.

This completes the illustration of the DNA strand of an organism from the outline. There is however two other things in the outline we need to map for step 1 to be complete. First, the interpreter must be defined, but we find quickly we don't know what to interpret – there is no problem to respond to – yet. It will have to wait until later when more information is gathered on the specific example problem given in a later section. Second, the environment has to be defined.

Stated more implicitly in the outline is the environment (or input) to define, or the part of the environment to pay attention to. These implicit details are one of the places the devil lurks. Are the environmental inputs words, sentences, shapes, etc.? It likewise will have to wait until a specific problem is given. It's the same situation the interpreter had. This lack of information to define these parts tells us something important. The Environment (Inputs) and the Interpreter need to be replaced for every different problem encountered, making them changeable parts. There are a few other similar problem specific elements to go on the changeable parts list. Altogether, they are the 1) Inputs, 2) DNA Instructions, 3) Interpreter, 4) Goals, and the 5) Outputs. These elements all need to be implemented differently for each unique problem.

Reproduction can also be done many different ways. It is reasonable to use a cumulative distribution function for the selection of parents, stochastic crossover, and 5% mutation rate for each bit position in the gene; these are as good as any other methods to use. These genetic operators are candidates to be the sixth part on the changeable parts list. They will be static here for simplicity of keeping the changeable parts list down to five items.

The other remaining steps can be mapped. Mapping the rest of the steps outlined above to something closer to computer language yields the following algorithm. Notice, this step would not be necessary if a universal adaptable GA was available as proposed. For now there is no universal GA, so the following steps are an idea about how to map the steps given earlier as a main function for the universal adaptable GA. These steps are the fixed parts to the Universal Adaptable GA.

Algorithm 1 Universal Adaptable GA Main

```
Input: Environmental inputs, initial chromosomes.
Output: Highest rated individual.
1: repeat for number of GENERATIONS (e.g., 2)
2:   repeat for number of ENV INPUTS (e.g., 14 shapes)
3:     repeat for number of CHROMOSOMES (e.g., 10)
4:       interpret (chromosome, environmental input)
5:   rate chromosomes
6:   if generation < GENERATIONS
7:     reproduce
8. save highest individual
```

If the individual is to perform in an actual real life environment, there needs to be another small addition made after the outline so the individual can perform in the environment, as shown below.

Algorithm 2 Universal Adaptable GA Perform

```
Input: Environmental inputs, chromosome.
Output: Response to problem.
1:  repeat for number of ENV_INPUTS (e.g., 14 shapes)
2:    interpret_real_life
        (chromosome, real environmental input)
3:  rate chromosome
```

Everything is mapped besides the changeable parts. Actually, these are the hardest parts to get right to make use of genetics. Before defining these parts, we need a problem to solve. In order to get something more concrete, an example problem is given.

4 Example Problem

What we have chosen for our fist problem to solve is the fairly well-known problem of dropping dies (or parts) on a cloth to make seat covers. The problem has many varieties. All varieties of the problem generally try to fit shapes on a surface, without overlapping, trying to get the least amount of waste as possible. You could add as many constraints to the problem as you like. You have to stop somewhere to get something done. The point being, the added constraints you impose should be implementable within a day's work. This is the right level of generalization without having to start over each time with a fresh implementation whenever presented with a new constraint, or problem.

4.1 Tetris Variant

We look at most hard problems as games. Instead of looking at the problem as cloth cutting, it can be fitted into something more entertaining, into a variant of the game of Tetris, with the same inherent problems. See Figure 1 for an arrangement of how the Tetris shapes should be placed on the board (or cloth) in a compressed format.

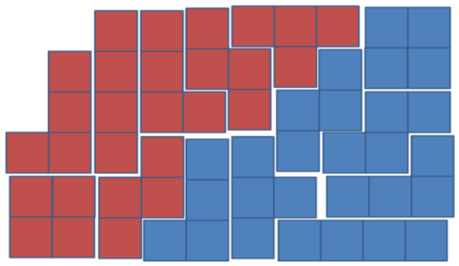


Figure 1: Tetris blocks arranged in compressed format.

The object of the "Tetris" game is to create an individual player, which places the different parts, the Tetris blocks called tetriminos, "tets" for short, on the board, opting to place the most pieces on the board as possible. It is easier to start with a smaller scale problem and solve the problem on this smaller scale before scaling it up to the same problem with larger size. You should eventually be able to scale it up to an inexhaustible size, and get a good enough solution using the same universal GA and adapting it to the larger board size with a little engineering help. Of course, first you have to understand how to solve the smaller scale problem before it will take a day's work to expand it to a larger scale problem. For our concerns the size of the board is a sheet of 6x10 squares. See Figure 2 for the board representation used.

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Figure 2: Inner board of 6x10 squares.

The border of four extra squares surrounding all sides of the inner board is to ease with implementation details, and seems to be a recurring way to make problems more implementable, instead of having to do level logic, such as, if this, else if that, else this, and so on. Exactly four outer board squares is constant for any sized board, for reasons shown later. It may have its genetic equivalence in biology as part of the extra bits material that doesn't seem to be used, in the non-coding regions of DNA, termed the dead code as opposed to the inner board, termed the active DNA or active code. It has definitely been useful when placing the proximity numbers on the smaller contained board, not having to worry about if the shapes go over the inner board's area.

4.2 Changeable Parts

The variant of the game of Tetris is the specific problem needed before being able to define the abstract parts into something concrete. The parts of GA on the changeable parts list, the 1) Inputs, 2) DNA Instructions, 3) Interpreter, 4) Goals, and 5) Outputs will now be defined.

5 Mapping Tetris Variant

5.1 Environmental Inputs

All shapes have four squares. There are a total of seven different types of tets. All types can be seen looking back to Figure 1: one of each tet shape in each symmetrical triangular group split by the diagonal. The actual environmental inputs will come in as shown in the following sequence.

square, s, z, T, L, J, l, square, s, z, T, L, J, l

Any amount of shapes which can cover at least 90% of the board space is reasonable to consider. Anything less is too trivial. There are 14 shapes, since 14 fills over 90% of the board. The tets could come down in random order, whatever the constraints they come down in; here they come down in fixed order. However elements you are sensing or suppressing usually have random behaviors.

5.2 DNA Instructions

The DNA strand (or genetic string) has been defined in a previous section, but without DNA instructions, DNA cannot do anything. Recall that DNA instructions are one of the different changeable parts for each implementation. Each gene has a different set of DNA instructions. The first DNA instructions that we define for the first gene are called proximity numbers. When the chromosome recognizes a shape, it categorizes it according to its DNA. Specifically, its gene value can be |00|, |01|, |10|, or |11|. The method each chromosome uses for proximity number instructions is decided by the chromosome's DNA by use of a case statement 0-3 with strategy names: TRIVIAL, SIMPLE, COMPLEX, MIXED (See Figure 3). There are a total of seven different tets. Strategy names are listed at the bottom. The proximity numbers keep the tets glued together by directing placement of next tets.

```
  Tet(0, 1)     Tet(1, 1)     Tet(2, 1)     Tet(3, 1)
(000000000)   (000000000)   (000000000)   (000000000)
(000000000)   (000000000)   (000000000)   (000000000)
(000000000)   (000000000)   (000000000)   (000000000)
(000000000)   (000001000)   (000011000)   (000001000)
(0000##000)   (0000##000)   (0001##100)   (0000##100)
(0000##000)   (0000##000)   (0001##100)   (0001##000)
(000000000)   (000000000)   (000011000)   (000010000)
(000000000)   (000000000)   (000000000)   (000000000)
(000000000)   (000000000)   (000000000)   (000000000)

  TRIVIAL       SIMPLE       COMPLEX        MIXED
```

Figure 2: Definition of the square tet. The # symbols represent shapes. The numerical values surrounding the shapes are proximity numbers.

The second set of DNA instructions for the second gene values to choose from is called Search Procedures (SP). One tricky part of studying DNA is in knowing the difference between the genes and the instructions carried out by the genes. For the second gene on the Tetris DNA strand, the |00| decides which SP the organism uses, meaning which set of instructions to carry out when searching for an open square to place the incoming tet. In other words, what set of DNA instructions does it select? Here there are only four different choices, SP0, SP1, SP2, and SP3; however, there can be more, but four is sufficient to gain an understanding. Notice there are 60 squares on each SP to match the size of the inner board. See Figure 4 for the four SP instructions serving as the second set of DNA Instructions for the DNA to select 0-3.

SP_0

1	6	15	29	37	38	30	16	7	2
5	22	28	42	48	55	49	43	23	8
14	27	36	47	54	59	56	50	31	17
13	26	35	46	53	60	57	51	32	18
12	21	25	41	45	58	52	44	24	9
4	11	20	34	40	39	33	19	10	3

SP_1

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60

SP_2

1	2	4	10	11	21	22	33	34	45
3	5	9	12	20	23	32	35	44	46
6	8	13	19	24	31	36	43	47	54
7	14	18	25	30	37	42	48	53	55
15	17	26	29	38	41	49	52	56	59
16	27	28	39	40	50	51	57	58	60

SP_3

11	16	46	37	30	52	9	39	20	4
29	35	2	15	24	56	47	3	33	27
43	59	49	38	31	14	41	26	44	54
17	12	8	23	1	32	10	53	58	18
7	55	51	40	19	60	34	21	13	48
22	28	5	6	45	57	25	50	42	36

Figure 4: Search Procedures

When following the DNA instructions for the second gene, the chromosome starts by trying to drop pieces on the inner board beginning at the corners with squares labeled 1, 2, 3, 4, and so on until 60, maybe moving in an inward pattern towards the center or maybe starting at a corner (1) and moving towards the other side on the diagonal if the DNA instruction specifies. The search procedures can be defined with any pattern, even random ones. The creator decides what to use for DNA instructions. The SPs in the figures are only examples.

5.3 Genes

It is important to notice there are only two genes |00|00| being actively used on the DNA strand. The other three genes are not being used for anything, although they could be. Some of the other organisms have some of their extra gene bits set to 1. Still they are not being used for anything at the moment. They may be activated for a different problem. Five total genes is a gross simplification of how DNA really works. Recall the mile high stack of procurement documents needed just to build the C-5 aircraft. Imagine the amount of information needed in DNA to build a human, or even a big toe. The DNA is doing much more than it first appears, and therefore should definitely be far longer than 10 bits in length.

5.4 Interpreter

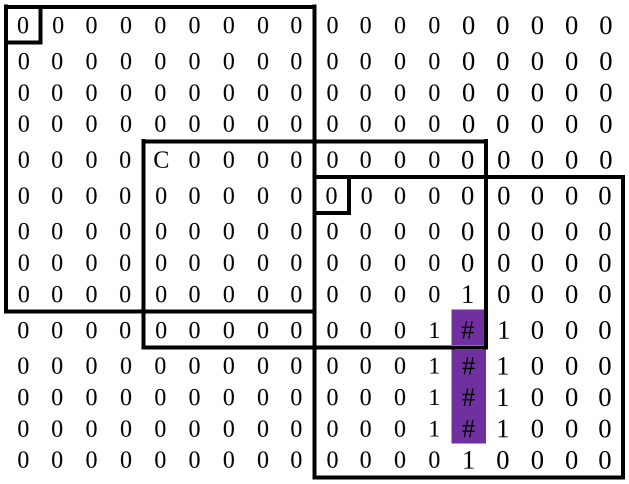
This is entering into interpreter territory, another one of the changeable parts to be made abstract. The interpreter is the key part to the GA. Despite the recent "solving of the genome" [4], with the many correlations found of genetic material to body characteristics, it appears that even with the DNA "source" code, we still do not comprehend exactly how this code is interpreted. We emphasize the interpreter, looking for a general purpose one, but once one is found, then the secret is figured out. For now all we have is the option to plug our dreams in there for how the interpreter works. The interpreter goes from the DNA strand to the full formed individual that works in the environment. Here, "interpreter" means roughly the same thing as it does in a language such as java. Something has to make the programs work. Something has to interpret the DNA strand to work in the environment. It is the problem of going from genotype to phenotype [5]. The cell in biology depends on the context it finds itself in. In Genetic Programming [6] the interpreter is done for us. Not everything can be written as nice little programs, and the GA is pulled off the shelf again to tackle harder problems. In the GA, this interpreter is difficult to define [1] for each problem. Since no one knows how it is exactly done in genetics, all ideas are just wild guesses and dreams. For us, the unknown has to point to some address in a universal GA, to where different interpreters can plug in as any genetic code interpreter you want. The interpreter interprets the given chromosome in the given environment to make the transformation into an individual that performs in the environment; in our case, a player that plays the variant game of Tetris. For an analogy, a book or blueprint is nothing without someone to read and interpret the book or interpret the blueprint to build the house. The perfect example to think about as an interpreter is the mother's womb. Somehow the mother's womb forms a baby out of the DNA instructions.

In our case there are only two pieces of data to interpret, the 1) Proximity Numbers, and the 2) SP. The interpreter, as was shown in the previous algorithms, takes the following form. Depending on the situation, there may need to be two interpreter(s) defined.

```
interpret (chromosome, environment)
interpret_real_life (chromosome, real environment)
```

5.5 Goals

The goal for this Tetris problem is fairly simple; to have the greatest number of tets placed on the board. Placing all 14 shapes on the board would be a perfect solution. There would be a minimum waste of only four squares. Goals are also related to constraints. The constraints for this problem are: shapes cannot overlap, and shapes have to be placed within the inner board area. If any portion besides the proximity numbers goes outside of the inner board's area, it is not a valid placement, and therefore is not placed. For example, the following in Figure 5 could not take place.



0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	C	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1	#	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1	#	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1	#	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1	#	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

Figure 5: Invalid Placement of Line Tet.

The line shape 'l' runs outside of the inner board. Now it can be seen why the square padding around the outside of the inner board needs to be size four. This is the minimum amount needed to contemplate placing the longest shape 'l' on the borders of the inner board. Anything larger would be wasteful, and anything shorter would not suffice. This contemplation is the farthest case where one square is on the board as close to the edge as possible. The two miniature squares are where placement starts and ends when contemplating placement. The 9x9 squares show the placement of the whole shape's area for proximity number. The square labeled 'C' is the center of another furthest possible proximity number to contemplate shape placement. Starting from the left top corner, using coordinates (1, 1) as the first miniature square on the corner extends to the (6, 10) second miniature square, spans the exact same size as the inner board.

The ranking computation sweeps over the entire inner board of chromosomes at the end of generations. For each '#' sign it adds 1,000 points. These are the places where the shape's area is placed on the board. The proximity numbers falling within the inner board are added on to this number to give the total score for the chromosome. The proximity numbers add when overlapped. For instance, placing 14 shapes on the 6x10 cloth would sum to 56,000 points (14 shapes x 4 squares per shape) + overlapping proximity numbers, say 15 more points, totaling 56,015 points, a perfect solution.

5.6 Outputs

The output is the last changeable part on the list. It is possibly more effective to understand the output by seeing a demonstration of the algorithm in action.

6 Demonstration

Even though the outline of the GA is not very complicated, it is helpful to be able to see everything sketched out at a detailed enough level to explain. So the demonstration uses this strategy to show the GA in process.

6.1 Mapping Outline

1) Problem: Place 14 tets on a 6x10 (60 square) board with the least amount of waste.
2) Tets come in to the cognitive system in the order: square,s, z, T, L, J, l, square, s, z, T, L, J, l.
3) One by one, shapes are perceived by each chromosome.
4) The chromosome recognizes the first shape as being a square. The interpreter translates what the |00| on the first chromosome's first gene means for the shape coming down. It means select the TRIVIAL strategy, or naked shape, meaning no proximity numbers around it.
5) Then, according to the chromosome's DNA value at the second gene, it selects one of the four search procedures to start trying to place the incoming tet. The range of places to try is from 1 to 60. If it cannot place the tet, it tries the next place in sequential order. Once it reaches 60, there is nothing more it can do for this shape. There is nowhere else to try to place the tet. Its board squares are all used up.
6) The above are done for two generations with 10 initial chromosomes. At the end of each *N*-1 generation, the individuals are ranked. Then parents are selected and offspring are reproduced. For the *N*th generation, selection and reproduction does not occur. The highest ranking individual from the last generation is saved to perform in the real environment.

The first attempt at placing 14 tets on a 6x10 board resulted in many isolated squares on the inner board. The isolated squares were greatly fragmented, not a good characteristic, and a low ranking best individual was produced, which took one hour to complete. The slow execution time is offset by being able to visually see and able to visually debug the GA in a spreadsheet G.U.I. and to allow quick and automated adjustments. There are a couple of things to try in order to correct generating these bad results.

Tet(2, 4)	Tet(2, 4)	Tet(2, 4)	Tet(2, 4)
(000000000)	(000000000)	(000000000)	(000000000)
(000000000)	(000000000)	(000010000)	(000000000)
(000000000)	(000000000)	(0003#1000)	(000000000)
(000010000)	(001110000)	(000##1000)	(00003#300)
(0001#3000)	(01###1000)	(0003#1000)	(0001###10)
(0001##000)	(003#30000)	(000010000)	(000011100)
(0001#3000)	(000000000)	(000000000)	(000000000)
(000010000)	(000000000)	(000000000)	(000000000)

Figure 6: Rotation of 'T' Shape with COMPLEX Strategy.

6.2 Corrections

The first idea is to add rotations. This took no more than a day's worth of programming work. See Figure 6 above for an example of one shape's rotation, the rotation of the 'T' shape for the COMPLEX strategy.

Something similar had to be done for the other six shapes and for all different strategies. Once the work was done, an unexpected and considerable speed-up in time occurred. Adding rotations sped up processing time significantly. The reason is because when trying each number from its SP on the board, there are more possibilities to try from the added three directions for open squares to drop the piece. However, the results were similar to the first attempt and not satisfactory. Something else needed to be applied to converge on a reasonable response.

The second idea was to add more search procedures. Another gene could be used or the alphabet of the current gene changed. The latter method was chosen. The problem was the gene only allowed for four different SPs to choose from, since the alphabet was binary. The idea was to convert from binary to trinary (00=0, 01=1, 02=2, 10=3, 11=4, 12=5, 20=6, 21=7, 22 = 8) to provide more search procedures to choose from without overwriting previous SP.

Applying trinary to the first gene on all the organisms yielded something similar to a modified definition of the DNA Strand. The trinary modifications can be seen on the DNA strands. Notice the remaining genes still use the binary alphabet.

Modified DNA Strands:	*Offspring DNA Strands*:
$organism_0$ = (0000000000)	$organism_0$ = (2100000000)
$organism_1$ = (0101010101)	$organism_1$ = (0001010101)
$organism_2$ = (0210101010)	$organism_2$ = (1011101010)
$organism_3$ = (1010000111)	$organism_3$ = (2210000111)
$organism_4$ = (1111001100)	$organism_4$ = (1110001100)
$organism_5$ = (1200110011)	$organism_5$ = (1000110011)
$organism_6$ = (2010101010)	$organism_6$ = (2001101010)
$organism_7$ = (2111111100)	$organism_7$ = (2000111100)
$organism_8$ = (1000000000)	$organism_8$ = (0200000000)
$organism_9$ = (2201010101)	$organism_9$ = (2211010101)

Increasing the first gene's alphabet to nine letters with trinary allows an extra possibility for each bit position (instead of only having two, this gives five extra spaces for five more search procedures) totaling nine search procedures to choose from, where there used to be only four. The best performing individual's SP after reproduction is shown in Figure 7.

SP_4

12	25	33	38	6	24	23	13	32	14
19	36	37	7	20	5	39	40	42	43
35	48	8	50	55	30	4	41	31	22
18	9	52	47	59	21	54	3	46	44
10	27	60	57	53	49	58	29	2	28
26	17	34	16	56	11	51	15	45	1

Figure 7: Best Performing Individual's SP.

The offspring of the individual resulted in the new DNA strands. Of these offspring, $organism_4$ performed the best, where it choose the first newly added fifth SP_4 and the COMPLEX strategy as its proximity number.

This combination of two changes resulted in a satisfactory solution. Only one shape from the environment did not make it onto the inner board area. This response is adequate for giving a minimal answer to the problem. The best individual produced was organism$_4$ = (1110001100) with ranking = 52,023. Output can be seen below in Figure 8.

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 3 0 0 0 5 0 2 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 3 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Figure 8: The Best individual solution produced.

7 Conclusions

Some areas of the DNA are well understood, and those should be the fixed areas in the universal adaptable GA. The fixed areas have already been figured out for the most part, and only need to be assembled into a main algorithm. These areas do not have to be reinvented every time a new problem is presented. The areas where there are always uncertainties are where there is always debate. These parts cannot be fixed at this time until they are more fully understood. Otherwise, the same basic GA will end up being reinvented over and over again by a new author for possibly the same problem yet all implementations would be done differently. These areas need to be abstracted away, left up to each unique realization. These ever-changing parts are the only things that need to change for each new problem encountered. The GA should be easily modifiable to satisfice many new problems without more than a day's worth of engineering work required for each new problem.

8 Future Work

The current implementation for SPs does not scale up to larger board sizes. For example, if the size of the inner board was increased to 50x50 squares, all SPs would have to be 2,500 cells. This would be infeasible, even with the number drag automation a spreadsheet provides. One idea is to change the DNA instructions; instead of SPs, use a command list, such as move left, right, up, down, choosing to place the tet as close to the left of the starting position as possible, which could be the closest right. This approach would make the individual more relative than absolute. Similar to how our hands are connected to our arms connected to our shoulders and neck relative to our head and so on.

The highest level goal in AGI is to build an AGI. This work has that goal in mind. The ultimate far reaching goal of this work is to disseminate the idea of learning how we can obtain intelligence from the brain or otherwise by means of GA. We already know the microbiology and the fundamental macro primitives of the brain; we only do not know how they all fit together. We understand why we have failed to learn how the brain works itself. The solution is to bring an outsider in, the computer, to crank away at the problem by giving it primitive functions, then using millions of years of evolution by speeding it up inside the computer to fit the known fundamental pieces together and generate new ideas via mutation.

If the A.I. went out to a rich environment such as the Internet, where there is constant change, read some text, and looked at pictures on a webpage, it should be able to be asked questions about what it perceived, and give an intelligent answer. Then we could declare with confidence we have figured out how to obtain intelligence. The idea is to use GA as control strategy to figure out how intelligence works, given sound primitives, such as NETL [8], hierarchy, lookup table, short term memory with 5+/-2 chunks, large matrix, production rules, an HTM for the five senses [9], even another GA itself (whatever you can dream up until we get it right) then letting GA work on it. All it takes is one person with the right idea.

Most of this research work was dedicated to solving a pseudo-Tetris problem with the GA as the learning mechanism to create the individual for playing the "game" well. In the background is this higher-level goal of creating a more general GA, to solve new problems never seen before.

9 References

[1] L. B. Booker, D. E. Goldberg, and J. H. Holland, "Classifier Systems and Genetic Algorithms"; Artificial Intelligence, 40, 1/3, 1989, 235—282.

[2] J. H. Holland, "Adaptation in Natural and Artificial Systems". The University of Michigan Press, 1975.

[3] J. H. Holland, K. J. Holyoak, R. E. Nisbett, and P. R. Thagard. "Induction: Processes of inference, learning, and discovery". MIT Press, 1986.

[4] J. C. Venter. "A Life Decoded: My Genome: My Life". Penguin Group, 1998.

[5] D. S. Burke, K. A. De Jong, J. J. Grefenstette, C. L. Ramsey, and A. S. Wu, "Putting More Genetics into GA", Evolutionary Computation, 6, 4, Oct 1998, 387—410.

[6] J. R. Koza, et al. "Genetic Programming IV: routine human-competitive machine Intelligence". Kluwer Academic Publishers, 2003.

[7] F. Gobet, et al. "Chunking Mechanisms in Human Learning"; Trends in Cognitive Science, 5, 6, 236—243.

[8] S. E. Fahlman. "NETL: A System for Representing and Using Real-World Knowledge". MIT Press, 1979.

[9] J. Hawkins, "Vision Framework Guide NuPIC 1.7.1" [Online]. Available: http://numenta.com
Oct 2011.