Applying Genetic Algorithms to Evolve Chess Piece Relative Values (April 2013)

G.K. Belmonte

***Abstract*— The relative piece values are one of the most important aspects of Chess AI, determining what a good move is from a lousy one. A Generic Algorithm was designed to verify piece values compared to the theoretical values given by chess AIs. The Genetic Algorithm fought individuals against each other, using a new kind of tournament based on Quick-sort as well as using common GA methods such as crossover and mutation over generations. The results often under-evaluated the Queen, but gave a somewhat close approximation of the other pieces, the closest given by 1.000,3.834,3.781,5.525,7.343, for Pawn, Knight, Bishop, Rook, Queen respectively. The work can be further refined with more execution time and improving parameters, though it does show to have promise.**

I. Introduction

CHESS playing algorithms are some of the oldest problems in computer science. Since the very beginning of computing, designing a computer algorithm capable of beating a chess-master has been one of the most interesting problems. IBM had been interested in this feat of strength since 1950, but it was not until Deep Blue came around, that a machine beat a Grand Master [1]. Creating a powerful computer capable did not only take the best hardware available at the time, but also the best algorithms designed by some of the strongest minds to utilize that power.

One of the aspects that one learns early in the game is that the value of the pieces is very different. And for the programmers, to evaluate the worth of a move, being able to assess what a "good move" was also a challenge. The ability to evaluate the worth of a move can thus make the difference between a good move and a great one.

For most cases, existing AI evaluate a position based on the following score:

Pawn	Knight	Bishop	Rook	Queen	King
1	3	3	5	9	∞

Fig. 1.1. A common standard material value table in chess. A knight is roughly worth 3 pawns and a queen just under 2 rooks. Also, losing a king loses the game, so it is worth all of the game.

These values are not entirely arbitrary. They were chosen based on the experience of experts [2]. A quick interpretation is that a computer would trade a knight for the worth of 3 pawns, if using the material value alone to determine the optimal move.

However, even experts disagree with the value of pieces, some attributing more value to some pieces than other, even giving individual pieces more value, such as the Queen's pawn.

Genetic Algorithms are fit to answer this dispute, optimizing over $\mathbb{R}^5$ to find those values that represent the value of the pieces, independently of arbitration.

II. Method

A genetic algorithm (here on GA) was implemented that would match the same AI fighting against itself with different relative material piece values. These individuals would compete over generations, will pass on their genes to create possibly better individuals that will themselves compete.

A. Individual Representation

Individuals are represented by 5 alleles, and one gene. Their phenotype is their performance in chess games. Due to the fact that the King remains to be the piece that cannot be lost, the genotype exists in $\mathbb{R}^5$, ignoring the King's value as to be really large. Individuals also have an assigned randomly generated name, to serve for quick identification purposes.

The initial population is generated with random alleles from integers ranging from 800 to 17000, both to allow a more accurate value for the pieces to be computed (e.g. Pawn-Value/1000 could be 0.950) as well as to fit with the AI algorithm more seamlessly. Furthermore, the values are not normalized, since normalizing on a pawn with a big value could cause other values to go well below the initial range, or inversely, well above it.

B. Fitness Evaluation

Fitness Evaluation is as always the most complicated. This is specially the case for this GA as the evaluation of the fitness of an individual is actually against the rest of the population. Many alternatives were considered.

Having the individual compete against the standard values was a possibility. This would mean the fitness would be how quickly it lost in terms of moves and score. This is due to the fact that an oscillatory behavior could occur, making move count actually meaningless. However, this approach presented a strength and two mayor flaws (for the purposes of the paper let N be the population size).

The strength was that the fitness evaluation would compute in $\Theta(n)$.

The first flaw was that it would not seek a global optima, but rather one that would beat the standard values.

The second flaw is that it would depend on the score of the game to determine how well it did, which would be computed using relative piece values, defeating the purpose, since it would optimize to maximize the score based on the standard piece values.

The next sets of approaches were inspired from the work of simulating organisms in an environment [3].

1. All against the best.
   Pros: Linear Time Complexity $\Theta(N)$
   Cons: Optimizing against the individual, not the actual target.
2. All against all
   Pros: Thorough evaluation of individuals' fitness
   Cons: Quadratic Time Complexity $\Theta(N^2)$.
3. Tournament
   Pros: Linear Time Complexity $\Theta(N)$
   Cons: All individuals matched against the best individual might be under evaluated.

The third option[1] showed the most promise, even if the original paper favored the first. The reason is that evaluating against the best individual had the disadvantage that a proper ranking would require a score on how well the individual did against the best. This got the GA back to square one, trying to understand how to compute the score. The third one was also unacceptable due to the fact that if by chance some of the better individuals were matched against the best, they would never be ranked appropriately.

The ranking was of extreme importance since the time to execute a single match was very large, so a new approach was derived inspired from existing algorithms. The approach number two and three gave great insight: a quadratic complexity algorithm could be reduced to linearithmic, much like sorting. Furthermore, for (2) each individual is tested against N-1 individuals, at least once, much like the naïve sorts. The last idea came from the Tournament match which resulted in a tree that would rank individuals based on depth, with depth $\log_2 N$. This was reminiscent of a max-heap.

Inspired from quick-sort came quick-tournament. From the population, and individual is picked at random, called the pivot. This individual is matched against the rest of the population, and this splits the population in roughly two parts. Those better and those worse that the pivot. The process is repeated recursively in the two halves of the population.

Since the execution of the match happens entirely asynchronously in the software, a special quick-tournament was designed so as to deal with call-backs as necessary. It should be noted that the quick-tournament evaluator can actually sort numbers[2], since it is designed on the very same principles of quick-sort, except that it is specialized for the purposes of this GA. The pivot always plays as black, and is taken randomly from the subset of the population. This guarantees that everyone has a reasonable chance at being ranked according to their skill, and furthermore, there is no need to sort the population according to score, since the algorithm intrinsically handles that, all in $\Theta(N \log_2 N)$.

The game length was restricted on a max-move basis and on a time-to-think per move basis. Different parameters were attempted. At the max-move threshold, the game is stopped and the player with the largest standard material score wins. This criteria was hard to pick, but was deemed acceptable due to the fact that at the point of termination, the game has advanced considerably, and all that is needed is an idea of who would win the game if it were left to run for longer. It is also expected to the values to evolve

[1] The analysis can be found in the appendix

[2] Even tested sorting numbers, since sorting a population is too expensive and not verifiable

near the theoretical values. And finally, the max-move parameter is set well above the average moves for a match, thus affecting the small subset of the matches and population.

A potential solution is to set the parameter of max-moves dynamically as some percentage above the average game of the last generation, so as to be able to evaluate properly most matches, while keeping a strict match termination criteria, with an absolute maximum if all matches are going on for too many turns, in which case it is likely that the population has near-equal fitness.

C. Parent Selection

Parents are selected on a probabilistic basis, based on the rank, zeroth being the lowest and $N-1$ being the highest. For each individual, a random number of parents is selected between 2 and 4 inclusive. When a parent is selected, his probability of going into the [2,4] parent pool is his rank divided by the population size. The probability is never one-hundred for an individual, and the bottom 20% are excluded from the selection.

$$\frac{0.2N}{N} \leq p(Parent) = \frac{rank}{N} \leq \frac{N-1}{N}$$

Eq. C.1. Probability of a parent being selected.

The parents are picked with replacement, so the same parent might be selected twice for the [2,4] pool. Each individual [2,4] pool will yield a single offspring. The number of offspring created is roughly 24%, on a probabilistic basis. This will be discussed in detail in the following section.

D. Genome Exploration and Exploitation

1. Crossover

Crossover happens on a per allele basis.

Originally, for each allele, the allele of a random parent of the [2,4] pool would be selected to be part of the offspring.

The following is an example of a crossover operation involving 3 parents and yielding a single offspring from the preliminary tests.

Parent 0

Pawn	Knight	Bishop	Rook	Queen
3000	9000	8000	4000	15000

+Parent 1

Pawn	Knight	Bishop	Rook	Queen
1860	2079	15692	15416	15736

+Parent 2

Pawn	Knight	Bishop	Rook	Queen
6110	12034	9677	1608	3273

= Offsrping

Pawn	Knight	Bishop	Rook	Queen
3000	2079	8000	1608	15736

Fig. 2.1. Original 3 parent crossover example

In preliminary results, it was observed that some genes came to dominate too quickly and the exploration was limited. For this reason, the allele was now passed only with a 2/3 probability, and one third as an arithmetic mean.

Parent 0

Pawn	Knight	Bishop	Rook	Queen
3000	9000	8000	4000	15000

+Parent 1

Pawn	Knight	Bishop	Rook	Queen
1860	2079	15692	15416	15736

+Parent 2

Pawn	Knight	Bishop	Rook	Queen
6110	12034	9677	1608	3273

= Offsrping

Pawn	Knight	Bishop	Rook	Queen
3000	2079	8000	1608	11336

Fig. 2.2. Updated 3 parent crossover example

2. Mutation

Mutation happens at an individual level, using simple simulated Gaussian Noise. Each allele has a chance to be mutated with Gaussian Noise with a standard deviation of 400 with a probability of 40%.

For further work, it should be considered to make the 5 dimensions of the standard deviation part of the individual itself, and allow them to evolve along with the individual.

E. Survivor Selection

The old generation is split into 3 parts.

The bottom 20% is thrown away, and replaced with completely newly generated individuals.

The middle 60% (20-80%) go through a probabilistic process.

There is a 20% chance that they will survive unchanged.

There is a 40% that they will be replaced by a mutant of the old generation, randomly picked with equal probability across the population, excluding the bottom 20%.

There is a 40% probability that they will be replaced by a created offspring, which they may or may not have passed genes unto, as explained in *C* earlier in this section.

The top 20% are kept intact, as by elitism.

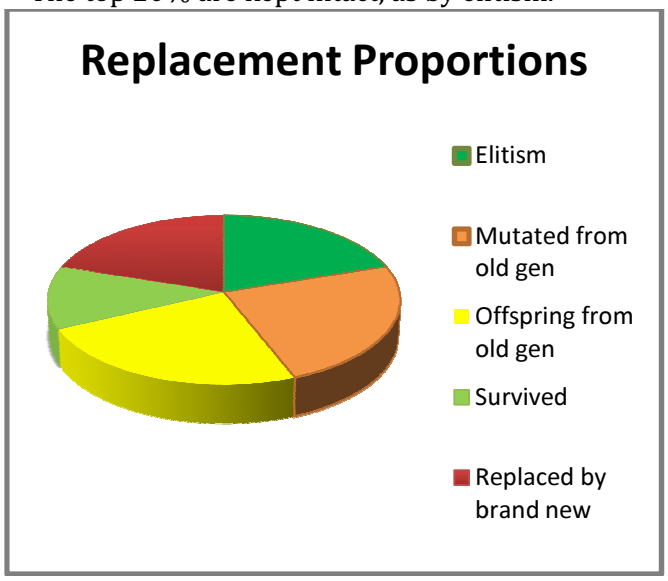

Fig. 2.3. Average replacement proportions. Since some of these are probabilistic, these could change from one generation to another.

F. Termination Criteria

The Termination Criteria for this Genetic Algorithm are not very creative. Given the huge amount of time to evaluate an individual, and thus a generation, the time restriction on this GA has to be huge for good results.

On the other hand, as it will be discussed in the results, genetic difference is harder to evaluate, since the GA will converge to a ratio of piece values, rather than a value. That is to say, 1:3:3:5:9, is equivalent to 2:6:6:10:18. So using a delta on the overall population to evaluate genetic stagnation is also not very useful. Fitness stagnation is also difficult to evaluate, since fitness is computed to respect to the population and does not have an optimal value of 0 or 1.

The number of draws will increase as the individuals are more and more fit towards the optima. This trend is analyzed, but not used in this GA. This GA is terminated simply on a generation cap.

G. Parameters

A small population is used to allow more generations to pass quickly, with about a third of a second thinking time for the AI. Given that the average length of the game is 50 moves, and a population of 12, using the information from the analysis of quick-tournament, the number of evaluations is roughly 31, this means 50 moves times 300 ms times 31, it takes some 10 minutes to evaluate a generation. For this reason, increasing the population on a single system is not reasonable over the time frame of this project.

H. Software

Four different chess AI's were considered: Ruffian, Crafty, Huo and Garbo. The first two were strong, but used book openings, meaning they could not be used, or would be too difficult to re-factor. The next one was relatively easy to re-factor, but played poorly and had documentation in Greek. The last played strong, written in JavaScript, thus it was easy to present a good user interface, given that every operation was rather negligible compared to the time for a match. Furthermore, it was perfectly cross-platform.

III. Results

A. Preliminary results

The results were very quick to converge around proportional values of some pieces. The average value of all pieces was roughly within the same staring point of halfway in between 800 and 17000 which is 8900. The two preliminary test sets, here on P1 and P2, had average piece value of 8664.4 $\pm$1519 and 7559 $\pm$ 1356 respectively (ave $\pm$ std.dev.). The following are the graphs of the pawn piece value for P1 and P2 with and without correction. The correction excludes the bottom 20% of the population on grounds that these could be mostly due to fresh individuals with huge differences in data with the rest of the population.

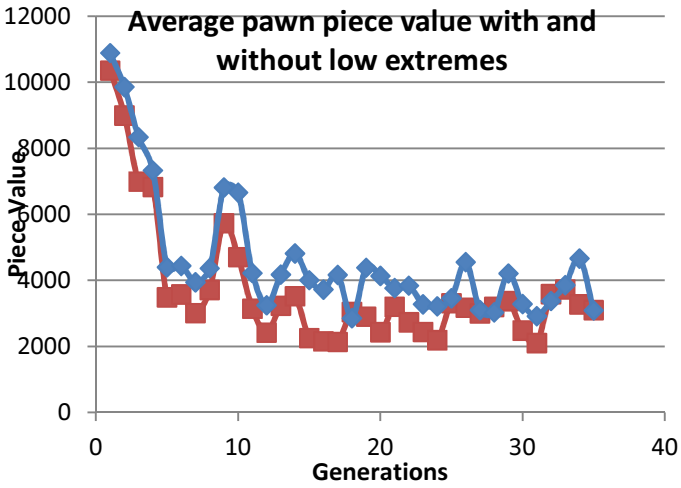

Fig. 3.1. P1 Pawn value over generations. In blue average of the population. In red, corrected average, excluding bottom 20%

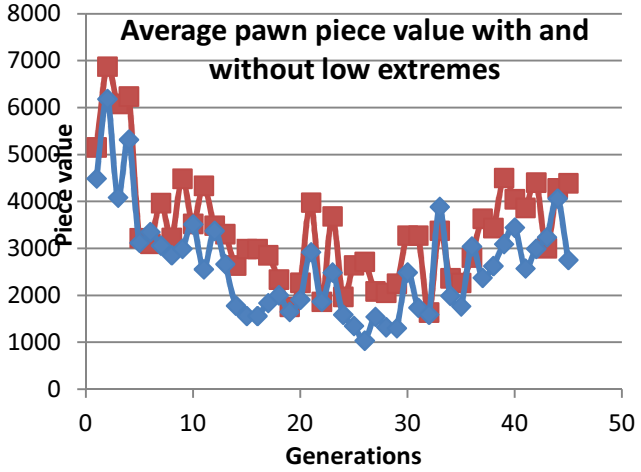

Fig. 3.2. P2 Pawn value over generations. In blue average of the population. In red, corrected average, excluding bottom 20%

It is especially clear in P1 that the corrected values are much lower and in fact, the standard deviation is also much smaller. Using this information, it is found reasonable to exempt the bottom 20% from the calculation of the generation average.

The following are the graphs for the piece values of P1 and P2 for all pieces over time.

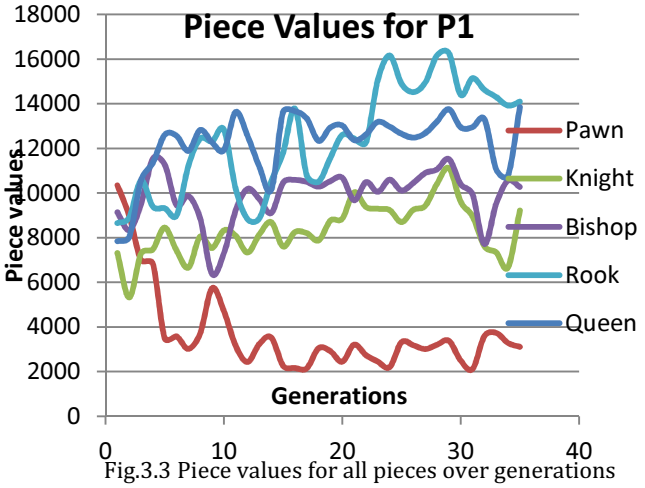

Fig.3.3 Piece values for all pieces over generations

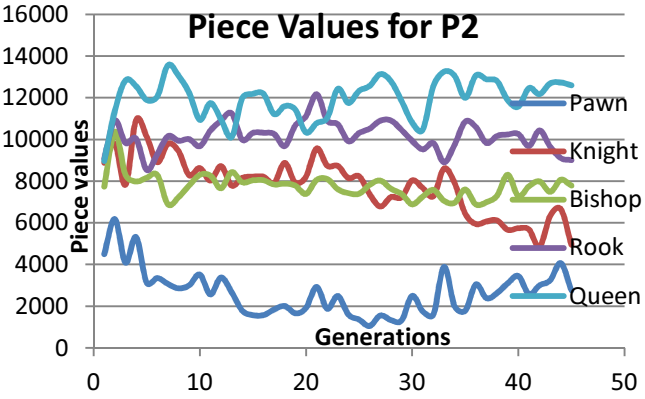

Fig.3.4 Piece values for all pieces over generations

The average values of the last 5 generations normalized over the value of the pawn are as follows:

	Pawn	Knight	Bishop	Rook	Queen
P1	1000	2520	3036	4561	3923
P2	1000	1820	2499	3063	4008

Table.3.1 Average values of the last 5 generations

Pawn average value was 3160 and 3127 for P1 and P2, respectively.

B. Secondary results

Using information from the results, the GA was tweaked to collect additional information, such as the number of evaluations per generation, and the number of draws. Using the same format as before, we have the following secondary results. S1 and S2 are the secondary results respectively.

Due to the huge variations in the draw counts, a moving average over 5 generations was taken.

S1:

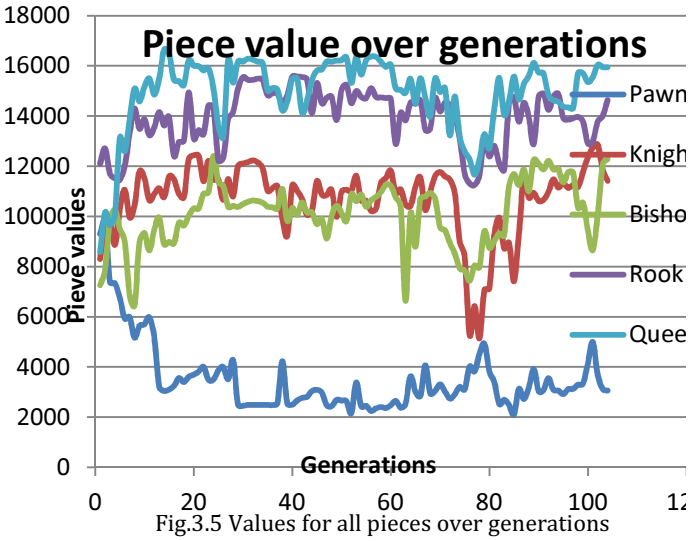

Fig.3.5 Values for all pieces over generations

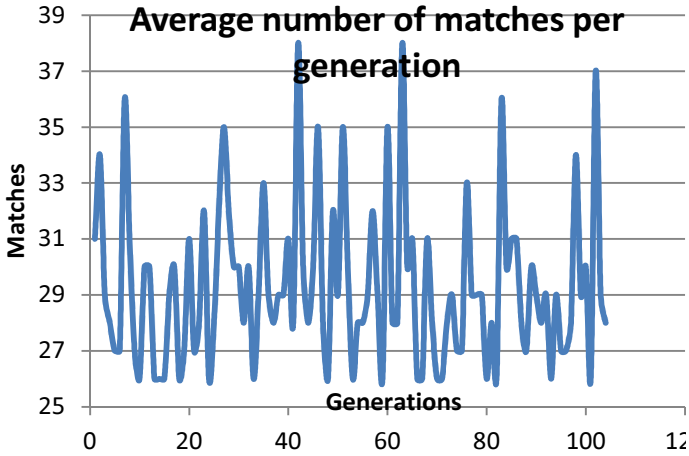

Fig.3.6 Number of matches over generations

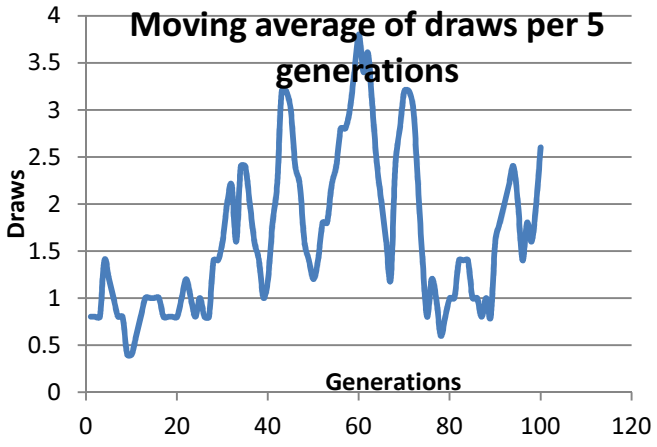

Fig.3.7 Draws over generations

S2:

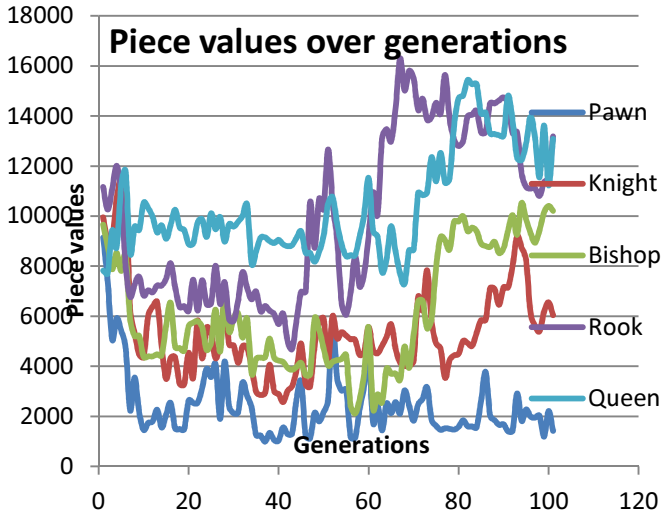

Fig.3.8 Piece values for all pieces over generations

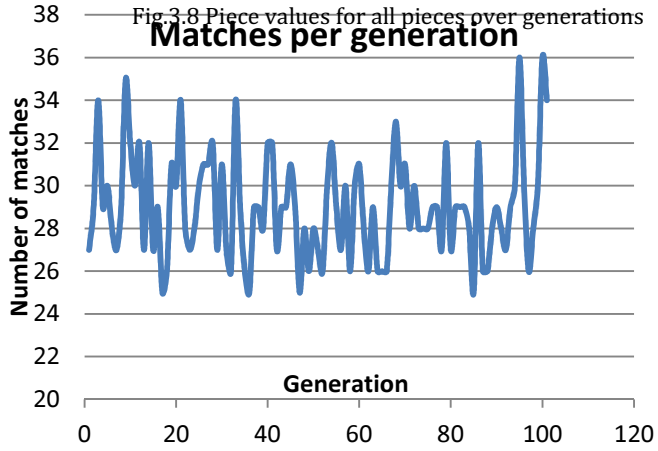

Fig.3.9 Number matches over generations

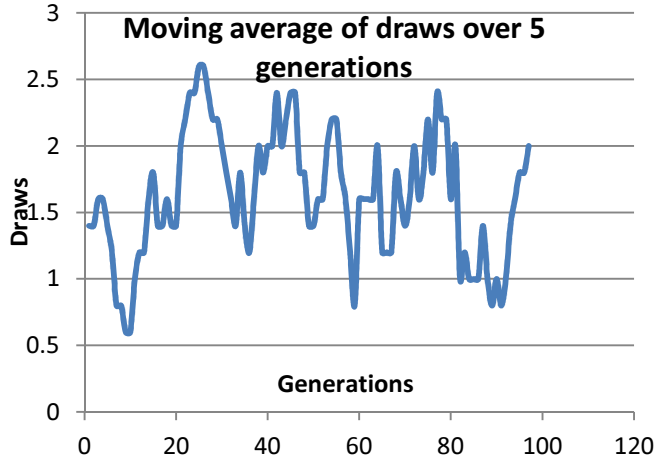

Fig.3.10 Draws over generations

A lot more information can be seen. First of all, we see more clearly how quickly the value of the pawn dies off. The value of the queen evolves just over the value of the rook, and the bishop seems to be worth just a notch over the knight. The oscillatory behavior seems to be strongly related. When a value goes high,

all of the other follow to reach proportion, while that value drops, maybe too low, making it oscillate back. The Rook of S2 shows this behavior at around 50 and 65 generations. As it drops, in both cases, the value of the queen increases with a lag until the rook value drops again. The system is heavily dependent on the values of others which causes pieces to often over-compensate for changes of their neighboring pieces. However, any dip or peak is quickly corrected by the GA over a generation or two.

Next, the number of matches oscillates wildly just around 30 for both cases. This confirms the complexity of quick-tournament by staying just above the hard low-limit of 25 evaluations per generation.

It is also very clear in S1 how the number of draws is slowly oscillating up. It isn't the case in S2 though.

The following are the normalized piece values of S1 and S2 for the last 5 generations

	Pawn	Knight	Bishop	Rook	Queen
S1	1000	3219	2759	3590	4145
S2	1000	3387	5585	6593	7116

Table.3.2 Average values of the last 5 generations

In this case, the average pawn value was 3804 for S1 and 1758 for S2. Remarkably, regardless of the very different piece values, the proportions get closer, especially for smaller pieces.

For the sake of seeing the best values, we present the last generation top 2 individuals.

	Pawn	Knight	Bishop	Rook	Queen
P1	2232	11628	10625	15054	13310
	2025	4904	10625	15054	14938
P2	2535	2279	8007	8707	10741
	1551	5535	8007	8793	13243
S1	3042	10150	12609	15456	15845
	3090	11720	12400	14975	15948
S2	1303	6099	10412	14389	14890
	1691	6099	10103	14389	10385

Table.3.3 Best two individuals for P1,P2,S1,S2

And below are the normalized values over the pawn:

	Pawn	Knight	Bishop	Rook	Queen
P1	1000	5210	4760	6745	5963
	1000	2422	5247	7434	7377
P2	1000	899	3159	3435	4237
	1000	3569	5162	5669	8538
S1	1000	3337	4145	5081	5209
	1000	3793	4013	4846	5161
S2	1000	4681	7991	11043	11427
	1000	3607	5975	8509	6141

Table.3.4 Normalized best two individuals for P1,P2,S1,S2

Immediately, we observe the Bishop having a higher value than the Knight which is worth about 3 Pawns, and a Rook and Queen living around 5 and 7 pawns. It is also possible to observe that for the most part, the proportions are conserved regardless of the actual Pawn values that we use to normalize to.

C. Final Results

The results were split into 4, each having a small change in the parameters. They will be named R1 through R4. R1 was used as a reference with the usual parameters a small population of 12 and thinking cap of 200 ms. R2 had a bigger population of 16. R3 was given twice the AI thinking cap, and R4 was given half, with a population of 15.

R1:

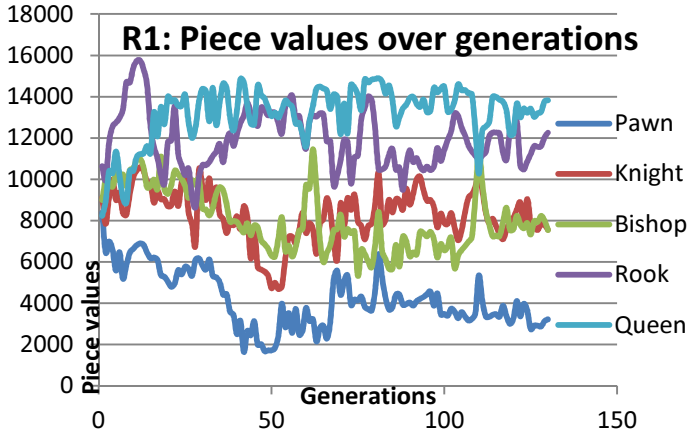

Fig.3.11 Piece values over generations

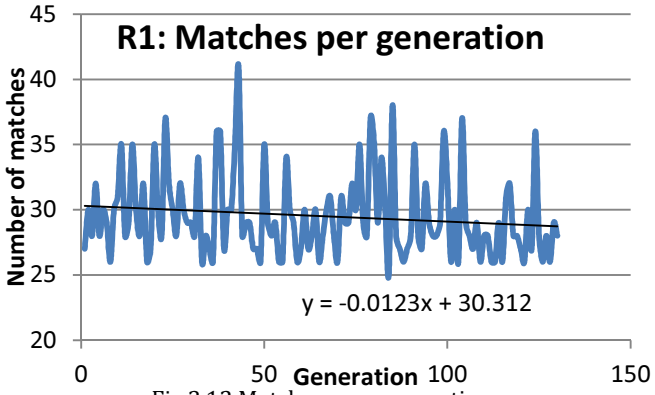

Fig.3.12 Matches over generations

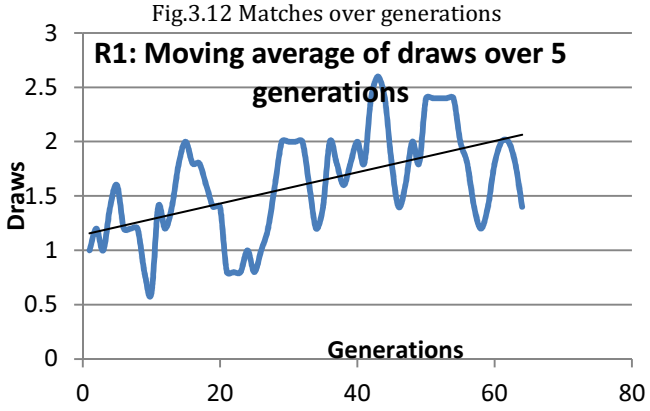

Fig.3.13 Draws over generations

R2:

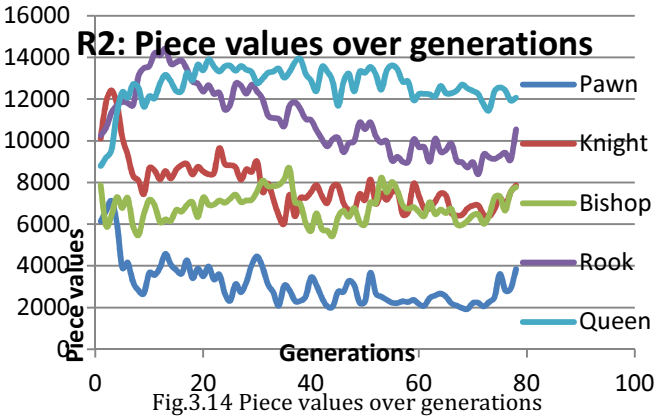

Fig.3.14 Piece values over generations

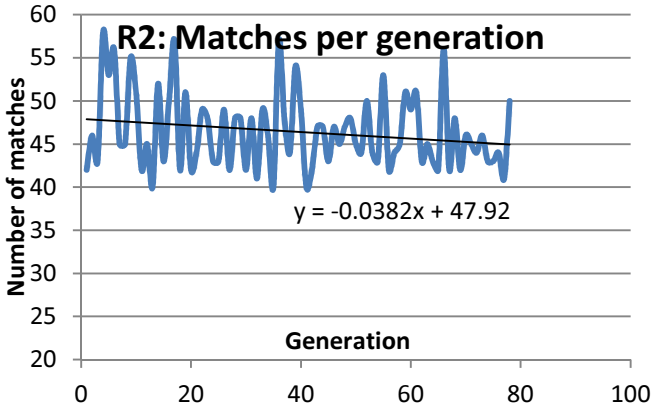

Fig.3.15 Matches over generations

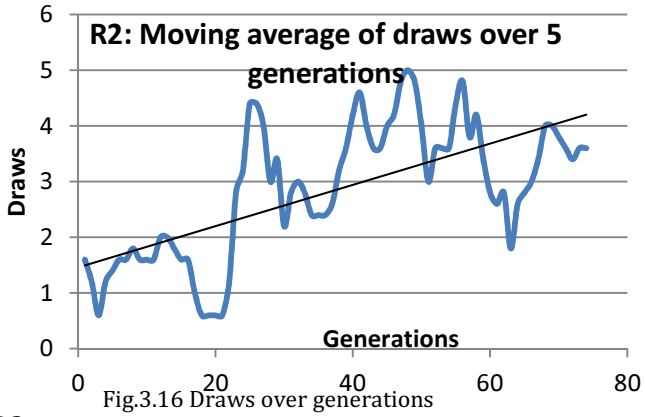

Fig.3.16 Draws over generations

R3:

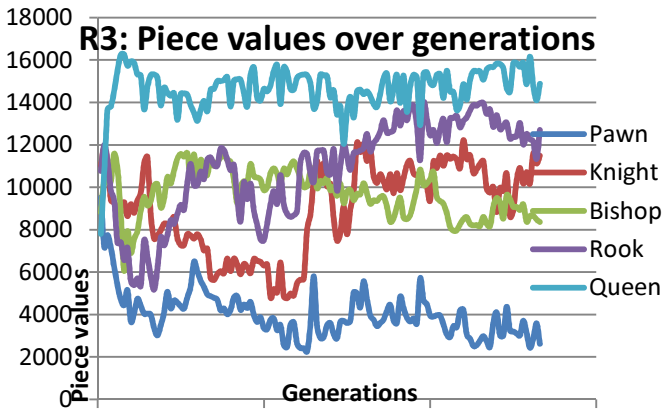

Fig.3.17 Piece values over generations

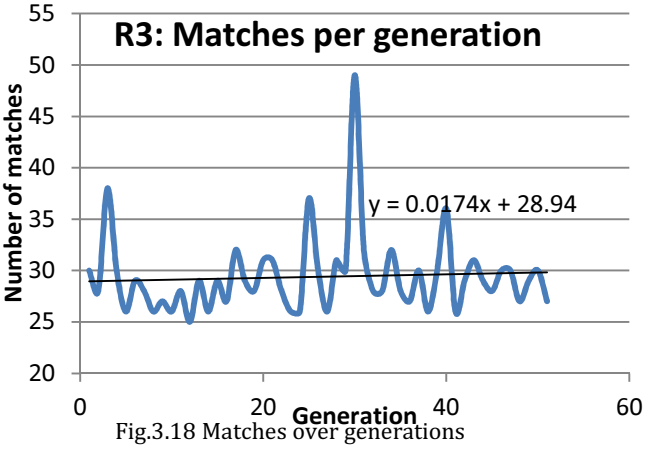

Fig.3.18 Matches over generations

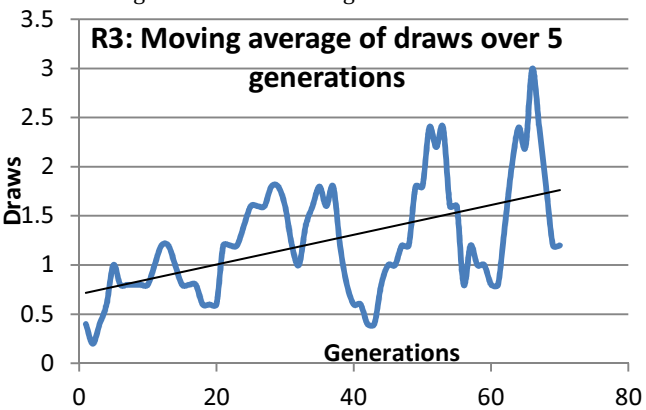

Fig.3.19 Draws over generations

R4:

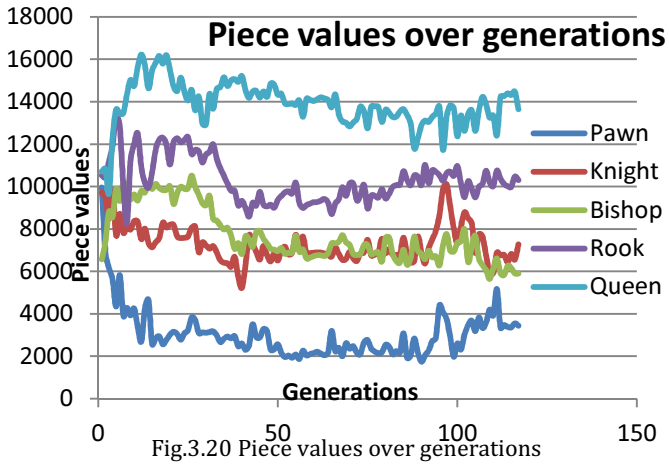

Fig.3.20 Piece values over generations

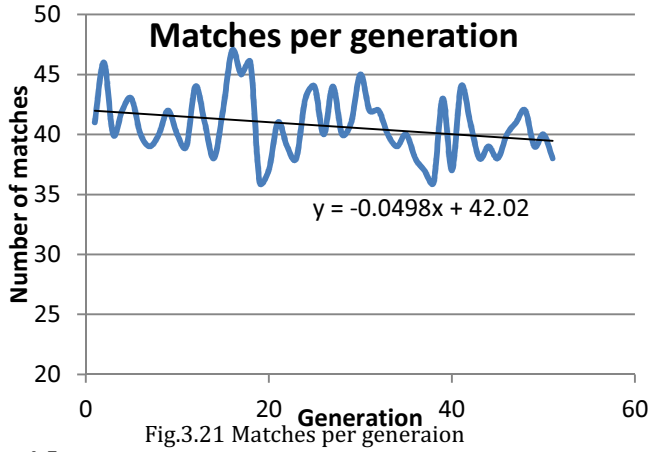

Fig.3.21 Matches per generaion

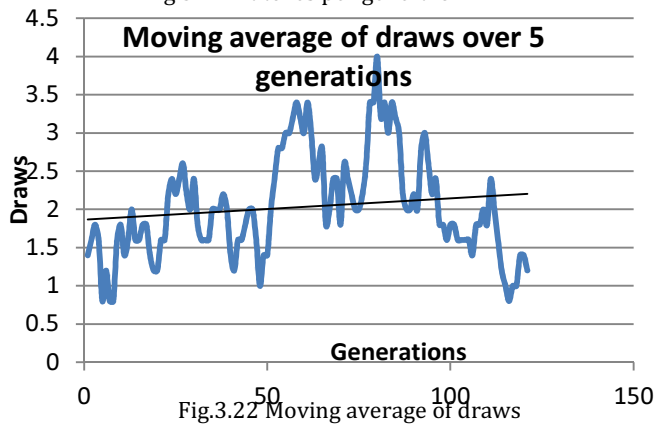

Fig.3.22 Moving average of draws

Finally, we present the elite:

	Pawn	Knight	Bishop	Rook	Queen
R1	2929	7243	7268	11930	13778
	2929	7432	7566	11421	13774
R2	1669	6399	6310	9222	12255
	1374	6364	6144	9222	12301
R3	2375	10531	8081	13296	16479
	2578	10331	8164	11618	16295
R4	2551	4643	10048	11585	15380
	2814	4731	10048	5478	15380

Table 3.5 Best individuals

Normalizing the elite

	Pawn	Knight	Bishop	Rook	Queen
R1	1000	2473	2481	4073	4704
	1000	2537	2583	3899	4703
R2	1000	3834	3781	5525	7343
	1000	4632	4472	6712	8953
R3	1000	4434	3403	5598	6939
	1000	4007	3167	4507	6321
R4	1000	1820	3939	4541	6029
	1000	1681	3571	1947	5466

Table 3.5 Best individuals normalized over pawn value

IV. Analysis

The set R2 with the highest population performed the best against the theoretical values, while R3 followed. R2 and R4, with high population seem to have the smoothest transitions, while R3, with a large thinking cap, had the worse. It may be because the extra time makes it easier for even weak values to survive.

As expected, the number of matches per generation follows accurately the prediction of quick-tournament, and the number of draws increases in all cases over time, accounting the noisy behavior.

The GA strangely behaves like a control system with external disturbance, noisily oscillating around a steady state value and over-compensating for jumps in the other piece's values.

It is interesting to see that many executions under-evaluate the Queen compared to theoretical values. This could either mean the GA gets stuck in a local optima, or that theoretical values over evaluate the queen. In either case, the result is very interesting, since no such other optima seems to have been suggested [2], or it could be possible to evolve a strategy involving sacrificing the Queen for enough material that could still make the game.

V. Further Work

The first consideration is that this GA needs massive amounts of time to reach some stability, thus many more generations are needed to see a final result.

Next, the use of piece values over time, as to see how the values of the piece evolve over a single game could be considered. In other words, attribute different piece values to the knight early game, as well as late game, and see how they differ.

By writing this GA, I discovered a lot of other semi-arbitrary parameters that could be interesting to evolve. One of these is a positional gradient that is applied to the piece depending on its position on the board. This favors Pawns that control the center as somewhat more important than a Rook that does nothing.

0	0	0	0	0	0	0	0
-25	105	135	270	270	135	105	-25
-80	0	30	176	176	30	0	-80
-85	-5	25	175	175	25	-5	-85
-90	-10	20	125	125	20	-10	-90
-95	-15	15	75	75	15	-15	-95
-100	-20	10	70	70	10	-20	-100
0	0	0	0	0	0	0	0

Table.5.1. Possible positional gradient for pawn.

The scalability of Quick-Tournament is really powerful, since it runs in quasi-linear time, and once a pivot has completed splitting a population in 2, the two halves can be evaluated in parallel, independently of one another, allowing multi-processor and parallel programming to be easily and quickly implemented. Given that this can run on any machine, scaling this would be really interesting, especially with the added explorations mentioned

Furthermore, adding a standard deviation to the genome to evolve it along with the individual could improve the GA.

Another possibility is pinning one or more pieces and observing how other pieces evolve within those constraints.

Finally, due to the dependency of the system variables with respect to one another, it seems like it over-compensates on changes. Studying this effect in other similar GA's and how to counter it could be of interest, perhaps treating it as an actual control system.

VI. Conclusion

The GA did not converge very close to the values expected, and it would take a very long time to reach high precision on the piece values for all pieces, however the ability for it to quickly move towards the neighborhood of an optima was very impressive, and many other results were obtained from it, such as the development of Quick-tournament, observing the behavior of draws across time, and the possibility of non-theoretical optima existing in the solution space. The GA is a work in progress and can definitely be improved as mentioned in the previous section in many different and interesting alternatives.

References

[1] Deep Blue [Online]. Avalaible: http://www-03.ibm.com/ibm/history/ibm100/us/en/icons/deepblue/

[2] Point Value [Online]. Avalaible: http://chessprogramming.wikispaces.com/Point+Value

[3] K. Sims, "Evolving 3D Morphology and Behaviour by Competition," Thinking Machine Corp., Cambridge, MA,Rep. 1994

Appendix I: Analyzing conventional tournament.

The conventional tournament generally occurs by taking two random individuals and matching them into one another and having the winners match against one another in pairs, until a champion is reached.

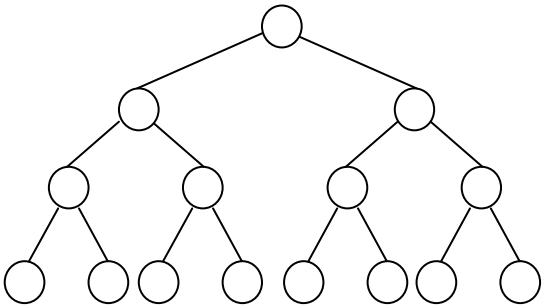

Fig.I.1 Example tournament scheme with 8 individuals

This means it takes $N/2$ evaluations to do the first level of the tree, and then $N/4$ to do the next.
Evidently when there is only one evaluation left, namely when $N/2^l$ is equal to one; no more evaluations will be computed. In this equation, l represents the level of the tree. It follows that:

$$N = 2^l$$
$$\log_2 N = l$$

and thus the number of total levels is $\log_2 N$ or in this case 3, for 8 individuals. In total the number of evaluations can be computed as

$$E = \sum_{i=1}^{\log_2 N} \frac{N}{2^i}$$
$$= N \sum_{i=1}^{\log_2 N} \frac{1}{2^i} = N\left(\frac{1 - \frac{1}{2^{\log_2 N}}}{2-1}\right)$$
$$= N(1 - 1/N) = N - 1$$

This method is of linear complexity.

Appendix II: Analyzing quick-tournament.

Quick tournament is based out of quick sort. The first step is to pick a random pivot individual, and match it against every other opponent.

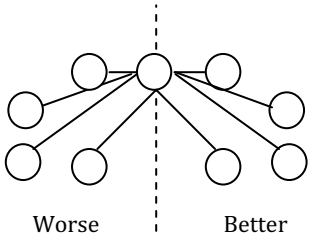

Fig.II.1 Quick-tournament split

Then we can split the participants into better or worse individuals, as compared to the pivot. It is then possible to use recursion on the new 2 populations.

Since each population is of size $\frac{N-1}{2}$, it will take each time half of the operations to complete it. Similarly, like the tournament, the depth is $\log_2 N$.

For the following section the general will be done in N and the example will be run with $N = 11$in squared parenthesis.

Assuming a perfect split and a population of N [11] the evaluations needed are:

First round:

$$N - 1\ [10]$$

Second round (one time each half of the population):

$$2\left(\frac{N-1}{2} - 1\right)\ [2(5-1)]$$
$$= N - 1 - 2\ [8]$$

Third round:

$$4\left(\frac{\frac{N-1}{2} - 1}{2} - 1\right)[4(1)]$$
$$(N - 1 - 2 - 4)[4]$$

Fourth round:

$$8\left(\frac{N-1-2-4}{8} - 1\right)[8(-0.5)]$$
$$(N - 1 - 2 - 4 - 8)[-4]$$

The Fourth round is in fact, not useful for our example anymore; no more evaluations are being performed.

k-th round

$$N - (2^k - 1)$$

This is valid so long as there are positive evaluations to perform. Thus,

$$N - \left(2^k - 1\right) > 0$$
$$N > 2^k - 1$$

Since the logarithmic is monotonically increasing for real numbers, we can take the log of both sides

$$\log_2(N) > \log_2 2^{k-1}$$
$$\log_2(N) > k - 1$$

which confirms our hypothesis that only $\log_2 N$ rounds have to be done. ($\log_2(11) = 3.45$, so 3 rounds).

The total number of evaluations is then

$$\begin{aligned} E &= N - 1 + (N - 3) + (N - 7) + \cdots \\ &\quad + \left(N - \left(2^{k-1} - 1\right)\right) \\ &= N - 1 + (N - 3) + (N - 7) + \cdots \\ &\quad + \left(N - \left(2^{\log_2(N)} - 1\right)\right) \\ &= N - 1 + (N - 3) + (N - 7) + \cdots + \left(N - (N - 1)\right) \\ &= N - 1 + (N - 3) + (N - 7) + \cdots + (1) \end{aligned}$$

$$E = \sum_{i=1}^{\log_2 N} N - \left(2^i - 1\right)$$

or

$$E = N(\log_2(N)) + \log_2 N - 2(N - 1)$$

This is of linearithmic complexity, given well picked pivots of course, this is the lower bound.
(N.B. For N = 12, E is 25).

Being less optimistic, we use the following, for an average execution count.

$$E = N \log_2 N - N$$

as an estimate.

Here are some key values compared to an all-against-all $\frac{N(N-1)}{2}$ tournament:

N	Estimate	Lower Bound	Full tournament
12	31	25	66
15	43	35	105
16	48	38	120
100	564	474	4950
1000	8965	7978	499500

Table.II.1 Comparison between Quick-tournament and Full tournament.