A Co-Evolutionary Approach towards Ms. Pac-Man vs. Ghosts Competetion

Kotikalapudi Raghavendra
rkyvb@mail.mst.edu

***Abstract* — Ms. Pac-Man is a challenging, classic arcade game with a certain cult status. From a computational intelligence perspective, Ms. Pac-Man is a challenging and interesting game as it captures competition and cooperation in a single game. In this report, we describe the use of competitive coevolution to evolve position evaluators for Ms. Pac-Man and Ghosts. Position evaluators are encoded as expression trees and are evolved using genetic programming. This report investigates the effect of CoEA/GP parameters on the gameplay and convergence of the algorithm. Some discussion and computational results are presented.**

I. Introduction

There has been interest in designing computer algorithms to play common games since the early advent of the modern digital computer. In this regard, chess has received the most attention, with Deep Blue beating Gary Kasparov in 1997 [1]. A major breakthrough occurred when backgammon AI achieved super human level of expertise solely by playing against itself and learning from the results [2]. With zero knowledge built in at the start of learning, this was an impressive feat. Other noteworthy successes include the use of coevolution to find a good position evaluator function for checkers [3].

Pac-Man is a classic arcade game originally developed by Toru Iwatani for the Namco Company in 1980. From a computation intelligence perspective, there are several aspects worthy of study. Pac-Man competes against a set of ghosts, which in turn cooperate with each other to trap Pac-Man, thus capturing competition and cooperation simultaneously in a single game. It'd be interesting to study how Pac-Man evolves a strategy against Ghosts and vice-versa.

Earliest work in this domain was that of Koza [4]. Koza used Genetic Programming (GP) to evolve Pacman position evaluator against predefined ghost behaviors [4]. Even though Pac-Man evolved a decent strategy, it was bounded by the fact that Ghosts used a predefined strategy of pursuing Pac-Man within the defined line of sight, coupled with occasional reversals to exhibit random behavior.

Gallagher and Ryan [5] evolved a Pac-Man player based on a finite-state-machine plus rule-set. In their approach they evolved the parameters of the rule set (85 in total), where the individual rules were hand-specified. However, the game simulator they used was a greatly simplified version of the original. Although the maze was a faithful reproduction of the original Pac-Man maze, only one ghost was used (instead of the usual 4), and no power pills were used, which misses one of the main scoring opportunities of the game.

De Bonet and Stouffer [6] describe a reinforcement learning approach to simple mazes that only involved one ghost. Even though their system learned basic pill pursuit and ghost avoidance behaviors, it removed cooperation aspect from Ms. Pac-Man.

More recently, Simon Lucas developed an open source JAVA framework [7] to capture all subtleties involved in the original Ms. Pacman game. This has already been the subject of on-going series of competitions since 2005. In [8], he describes the use of Evolutionary strategies to evolve a neural network position evaluator against a more realistic set of Ghost behaviors. The evolved Pac-Man controller exhibited surprising traits that are otherwise not obvious to a human designer. One such trait was that the evolved Pac-Man controller exhibited a strong attraction towards the strongest Ghost and power pills. The reason for this is that fact that it enables Pac-Man to lure the ghost to a power-pill and subsequently eat the ghost.

In this report, we study the possibility of using co-evolutionary GP to simultaneously evolve position evaluators for Ms. Pacman and Ghosts, thereby preserving competition and cooperation aspects of the game. Section II describes the implementation details along with design decisions of CoEA/GP. Section III describes the experimental setup along with the choice of CoEA/GP parameters, followed by results in section IV and discussion in section V. Finally, a conclusion is presented towards the end of this report.

II. Methodology

We used the open source JAVA simulator [7] by Simon Lucas for its object oriented nature and ease of use. Their implementation is a reasonable approximation of the original game with minor exceptions as described in [8]. The simulator is primarily used to evaluate fitness of Ms. Pacman Controller. The remainder of this section describes the exact details of how a GP and CoEA framework was implemented.

A. *Representation*

The controllers for Ms. Pacman and ghosts are based on position evaluation. At every state, the set of possible moves are enumerated and the one with highest score is chosen as the new state. To keep the framework simple and computationally efficient, generation of game tree and minimax search is avoided.

Position evaluators for Ms. Pacman and Ghosts are represented as expression trees. There are four Ghosts in the game. Instead of using a different position evaluator for each ghost, a single evaluator is used. This was done primarily to meet time and computational constraints of the project. In order to preserve cooperation aspect among the ghosts, we included a terminal node which determines the Manhattan distance to the nearest ghost. Although simple, it provides the means to evolve a cooperative strategy among the ghosts while providing the needed distinction among the four ghosts.

The function set is common to both the evaluators. It includes basic arithmetic operators such as {+, -, /, *}, along with a RAND (a, b) node that generates a uniform random number between [a, b]. RAND node adds unpredictability and non-determinism to the controllers.

The terminal set for Ms. Pacman and Ghosts is shown in Table I and Table II respectively.

TABLE I. TERMINAL SET FOR MS. PACMAN

Terminal Node	**Description**
DoubleConstant	A constant double value
NearestManhattanDistPacmanGhost	The Manhattan distance of Ms. Pacman to the nearest ghost
NearestManhattanDistPacmanPill	The Manhattan distance of Ms. Pacman to the nearest pill

TABLE II. TERMINAL SET FOR GHOSTS

Terminal Node	**Description**
DoubleConstant	A constant double value
ManhattanDistPacmanGhost	The Manhattan distance between Ghost and Ms. Pacman
NearestManhattanDistGhost	The Manhattan distance between of the given ghost to the nearest ghost

There are undoubtedly several others that could have been included. Instead, for the preliminary study, we decided to keep it simple.

B. Random Number Generation

We used the "Random" class in JAVA for generating all the random numbers. With the same seed and sequence of method calls, it will generate and return identical sequences of numbers. This enables us to reproduce all the experiments.

C. GP Methodology

Genetic programming (GP) is a special variant of genetic algorithms (GA) where an individual is represented as a computer program. Typically, expression trees are used to represent the program. Each stage of GP in context to Ms. Pac-Man is described below:

i. *Initialization:* The initial population consists of randomly generated expression trees using ramped half and half method. In ramped half and half method, a maximum depth D_{max} is chosen and each individual is created with Full or Grow method, chosen with equal probability. In the full method, each branch of the tree is allowed to reach a depth D_{max}. The contents of node at depth d is chosen from the function set (FS) if $d < D_{max}$ and from the terminal set (TS) if $d = D_{max}$. In the grow method, each branch is allowed a depth upto D_{max}. This is done by stochastically choosing a node from FS ∪ TS while $d < D_{max}$.

ii. *Parent Selection:* Selection determines which individuals are chosen for mating. Typically, parent selection uses fitness proportionate selection. However, to maintain greater selective pressure, we use over-selection. In over selection, the population is first ranked by fitness and then divided into two groups, one containing top x%, with (100-x) % in the other. In our toolkit, 'x' is a configurable parameter.

iii. *Recombination:* Recombination produces new individuals by combining the information contained in the parents. Recombination in GP uses sub-tree crossover operator, which works by interchanging the sub-trees starting at two randomly selected nodes the parents. From the implementation perspective, deep copies of both individuals are made (we used an open source java library: http://robust-it.co.uk/clone/), followed by random selection of nodes in both parents. These nodes are swapped by modifying parent and child pointers.

iv. *Mutation:* Mutation introduces new genetic material by producing small random variations in the individual. In GP, we employed sub-tree mutation operator, which works by randomly changing the node type, or changing the value. When a node type is changed from functional to terminal node, all the children are trimmed off. In the opposite case, nodes are added until arity requirements of all the functional nodes are satisfied. To avoid infinitely long trees by this procedure, we upper bound the tree length by a configurable value. This was done to meet computation and time constraints on this project. Lastly, change of node value is only applicable to 'DoubleConstant' node. Mutation in this case tweaks the value by adding a random number generated between [-1, 1], thereby providing the means to increase or decrease in value.

v. *Survival Selection:* This stage provides the necessary evolutionary pressure for the solutions to evolve. No survival selection effectively reduces the algorithm to a random search. In our implementation, we adopted K-Tournament selection and truncation selection. Tournament selection involves running several "tournaments" among a few individuals chosen at random from the population. The winner of each tournament is added to the selection pool. Selection pressure is easily adjusted by changing the tournament size. If the tournament size is larger, weak individuals have a smaller chance to be selected. In truncation selection, candidate solutions are ordered by fitness and some proportion of the fittest individuals are selected. In all our experiments, we adopted K-Tournament selection.

vi. *Parsimony Pressure:* Parsimony pressure is implemented as a means of controlling bloat in GP. Bloat refers to the tendency of programs generated with GP to grow extremely large without corresponding increases in fitness. This phenomenon is well documented in the GP literature [4]. Most code growth consists of code that does not directly contribute to a program's performance. In our implementation, we penalize fitness based on the size (number of nodes) of the tree. i.e., Fitness = (Fitness due to solution quality) – P * size, with 'P' being a configurable value.

vii. *Termination:* Our termination condition is based on number of fitness evaluations.

viii. *Fitness:* In a GP only version, fitness of Ms. Pacman is defined as the score it obtains in a single game. A single game runs for a maximum of twice the number of grids in the maze with single life as opposed to three lives. There are 220 food pills, each worth 10 points and 4 Power Pills, each worth 50 points. The score for eating each ghost in succession immediately after a power pill starts at 200 and doubles each time. So, an optimally consumed power pill is worth 3050 (= 50 + 200 + 400 + 800 + 1600). Note that if a second power pill is consumed while some ghosts remain edible from the first power pill consumption, then the ghost score is reset to 200. In addition to these, fitness is boosted by the remaining time whenever a level is cleared successfully. This allows evolution of efficient controllers, and prevents games from going on indefinitely.

D. Competetive CoEA Methodology

Coevolution refers to a special form of evolutionary computation in which the fitness evaluation is based on interactions between multiple individuals. In competitive coevolution, increase in fitness of one population decreases the fitness of the other population. Competitive coevolution effectively leads to an "arms race" in which the two populations reciprocally drive one another to increasing levels of performance and complexity.

In our framework, we adopt a two population (or demes) based competitive coevolution among Ms. Pacman and Ghost. The fitness for Ms. Pacman is determined by the average of fitness scores obtained against a sample of ghost population. In our experiments, we use a sample size of 60%. Again, the reason for this is due to time and computational constraints.

The fitness of a ghost against a given Pacman individual is defined as (MaxPacmanScore – Pacman fitness). For our setup, it is safe to assign MaxPacmanScore = 50,000. Therefore, an increase in Ms. Pacman fitness causes a decrease in ghost fitness. Finally, fitness of a ghost is defined as the average of all scores obtained against Ms. Pacman population. For this study, Hall of fame and disengagement prevention mechanisms were not implemented.

III. Experimental Setup

We investigate two ways of evolving Ms. Pacman controller. In the first experiment, a random ghost controller is used and Ms. Pacman population is evolved using GP. Maze 1 and Maze 4 of Simon Lucas' JAVA framework is used, each with 30 runs for 2000 fitness evaluations. Detailed parameter setup for Ms. Pacman only evolution is shown in Table III. The object of this experiment is to study how maze difficulty affects the evolution of Ms. Pacman. Maze 4 is more difficult than Maze 1.

TABLE III. GP Setup for Ms. Pacman Only Evolution

Parameter	**Value**
Number of runs	30
Max evals/run	2,000
μ	40
λ	75
Mutation Rate	0.15
Recombination Rate	0.85
D_{max} for initialization	5
Parsimony penalty coefficient	8
Parent Selection	Over selection with x = 0.35
Survival Selection	K-Tournament selection with k = 4

For each run, a log file containing evaluations vs. fitness is maintained. Lastly, a solution file containing the expression tree for the fittest Pacman individual is logged. In addition to these, a world state sequence containing the sequence of state transitions for the best game is maintained.

For the second experiment, both Ms. Pacman and ghost is evolved using competitive co-evolutionary GP. In this experiment we study how parsimony pressure for Pacman population and ghost population affect the convergence and the quality of solutions. For the CoEA setup, parsimony pressure for each population is varied from the set {0.1, 0.25, 0.5}. Each of these resulting nine combinations is run 30 times with 3,000 fitness evaluations per run. Detailed parameter setup for CoEA framework is shown in Table IV.

TABLE IV. Parameter Selection For CoEA

Parameter	**Value**
Number of runs	30
Max evals/run	3,000
μ	20
λ	40
Mutation Rate	0.15
Recombination Rate	0.85
D_{max} for initialization	5
Parsimony penalty coefficient	3
Parent Selection	Over selection with x = {0.1, 0.25, 0.5}
Survival Selection	K-Tournament selection with k = 4

For the CoEA version, in addition to logs specified in GP version, we also log the expression tree for the best ghost individual. In both experiments the function set and terminal set for Ms. Pacman and Ghost population is as described in section 2-A.

IV. RESULTS

A. *Competetive CoEA Results*

As described in the experimental setup X% in over-selection is varied form the set {0.1, 0.25, 0.5} for both populations. Table V presents detailed statistics containing the best fitness value for each run. Each run consists of 3000 fitness evaluations.

TABLE V. BEST FITNESS VALUES FOUND BY COEA FOR (X_{PACMAN}, X_{GHOST})

Runs/(Xp, Xg)	**(0.1, 0.1)**	**(0.1, 0.25)**	**(0.1, 0.5)**	**(0.25, 0.1)**	**(0.25, 0.25)**	**(0.25, 0.5)**	**(0.5, 0.1)**	**(0.5, 0.25)**	**(0.5, 0.5)**
1	1074.167	1418.889	1006.944	1090.75	1309.722	1518.056	964.4444	935.2778	1643.333
2	858.3333	1511.5	1092.222	744.7222	6651.389	2339.917	991.1111	1289.444	1122.222
3	1618.222	986.6667	3171.306	1304.167	1235.25	1379.444	1095.556	1419.694	1118.333
4	954.1667	7824.528	1145.278	1544.583	1800	1275.278	1313.611	1318.333	1151.944
5	3441.583	1621.389	2427.333	1125.833	1450.278	894.1667	1074.167	1477.778	2490.917
6	1471.667	942.2222	1570.444	1918.333	1486.944	1507.222	831.6667	4332.639	1058.056
7	748.3333	1754.056	1231.667	1333.917	878.6111	790.2778	2090.222	1208.611	1940
8	2040.111	2556.222	1328.056	1611.944	978.6111	1573.722	975	1231.667	1501.111
9	1332.778	1341.944	858.8889	967.7778	2386.778	934.4444	1395.556	1891.917	1380.278
10	1195.278	990	2275.417	1045	1121.667	1190	1257.778	1716.778	1613.333
11	1222.5	1400	2999.833	1275.833	1516.667	1230	4587.639	1231.667	1092.778
12	5754.167	1256.667	923.3333	1627.778	1034.722	1445	841.6667	1400.556	1155
13	1162.5	850	993.6111	2500	1188.889	1318.889	1577.444	1669.333	1713.5
14	763.6111	1005.278	1301.667	6275	1362.778	1331.111	2032.667	756.9444	1147.778
15	1381.389	1535.333	1688.889	1699.222	1719.167	920	1039.167	2205.083	1080
16	1399.444	1313.333	2359.833	1244.444	1258.056	943.0556	1452.917	1346.111	4321.167
17	1379.167	1024.167	1019.167	1054.444	800	983.3333	1734.167	1221.111	1687.778
18	1303.056	904.7222	884.4444	2488.167	1448.333	1470.833	1165.556	1272.222	1293.611
19	951.6667	986.3889	1800	942.5	1546.111	1946.472	1603.028	1286.111	1790.778
20	1345.889	1174.722	1248.333	1026.389	2138.667	1315.556	832.5	1115.833	976.6667
21	1628.333	1400.944	2466.167	1396.667	1537	1550	1859.083	1503.056	1449.028
22	1437.5	2351.694	2770.333	1682.778	1129.167	1100	970.2778	1215	924.1667
23	1111.944	2631.944	1495.111	1236.667	1548.083	10896.83	958.6111	2922.833	1255.833
24	1372.5	1150.556	1608.611	781.6667	2246.944	1108.333	1096.389	1589.167	958.8889
25	1292.5	1322.222	1151.111	1928.528	5456.639	1307.222	1841.417	1020.278	1538.333
26	1269.222	1082.222	1216.111	1450.278	2670.667	5800	1239.444	2728.639	1139.167
27	1203.056	974.4444	1012.222	2596	1400	964.1667	1712.778	1117.5	3553.861
28	1024.167	2681.028	913.8889	991.3889	1236.667	804.1667	1350.278	1058.889	1267.5
29	1193.333	1400	1811.889	779.4444	1316.389	1390.833	1127.778	1331.722	2057.278
30	2396.25	1044.444	1200	1251.111	840.8333	1036.667	1242.778	1763.056	1139.167

To test the combination that yields the fastest convergence, instead of performing two-sample t-tests on 32 possible pairs, we used ANOVA as it reduces the chance of committing a type 1 error. Results of one way ANOVA with α = 0.05 is shown in Table VI and Table VII.

TABLE VI. SUMMARY OF STATISTICS FOR TABLE V

Groups	**Count**	**Sum**	**Average**	**Variance**
(0.1, 0.1)	30	45326.83	1510.894	909587.9
(0.1, 0.25)	30	48437.53	1614.584	1642183
(0.1, 0.5)	30	46972.11	1565.737	452549.8
(0.25, 0.1)	30	46915.33	1563.844	1035872
(0.25, 0.25)	30	52695.03	1756.501	1584939
(0.25, 0.5)	30	52265	1742.167	3783796
(0.5, 0.1)	30	42254.69	1408.49	490722.4
(0.5, 0.25)	30	46577.25	1552.575	499995
(0.5, 0.5)	30	46561.81	1552.06	562219.7

TABLE VII. RESULT OF ONE WAY ANNOVA WITH ALPHA = 0.05

Source of Variation	**SS**	**df**	**MS**	**F**	**P-value**	**F crit**
Between Groups	2837739	8	354717.3	0.291233	0.968522	1.973975
Within Groups	3.18E+08	261	1217985			
Total	3.21E+08	269				

Since *P-value* > *α* and F < F critical, we cannot reject the null hypothesis. Therefore, for this particular problem, at least on 3,000 evaluations, variation of 'X' in over-selection for both populations doesn't have a significant effect on convergence of CoEA. Fitness vs. Evaluations plot for all nine cases is shown in Figure 3.

B. *Ms. Pacman Only Evolution with GP*

Ms. Pac-Man was evolved against a random controller for Maze 1 and Maze 4 with parameters as specified in Table III. In both Mazes, Ms. Pac-Man achieved a maximum score of 32348 against the random controller. Fitness vs. Evaluations plots for the best run in Maze 1 and Maze 2 is shown in Figure 1. Detailed statistics are not presented due to time constraints of this assignment.

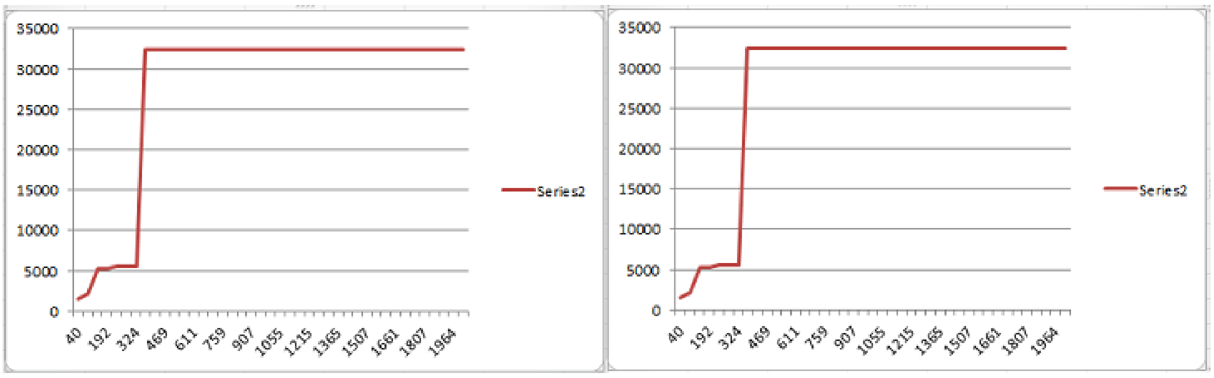

Figure 1: Fitness vs. Evaluations plot for the best run on Maze 1 and Maze 4 respectively

Finally, a random search algorithm is used to randomly control Ms. Pacman agent. Again, 2000 evaluations are used. The plots of Fitness vs. Evaluations for Maze 1 to Maze 4 are shown in figure 2. On average, random search achieved an average of 1500 on maze 4, which fades in comparison to GP or CoEA. Therefore, on this problem, Evolutionary search is definitely better than a random controller.

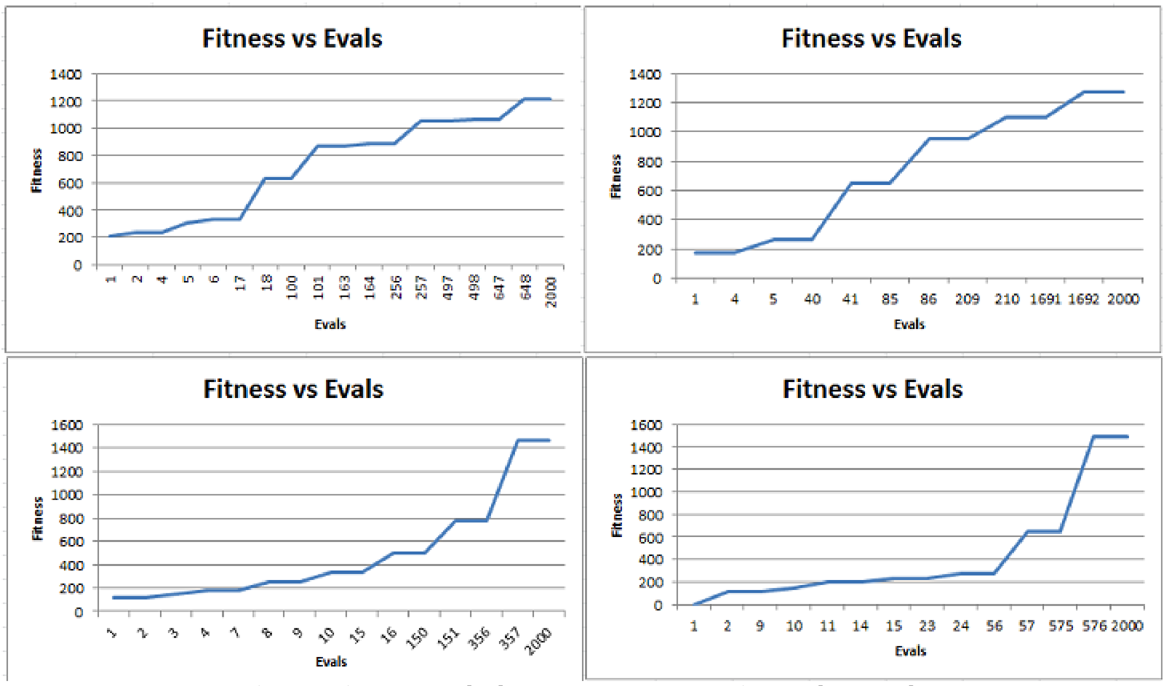

Figure 2: Fitness vs. Evals plot on Maze 1 to Maze 4 using Random Search

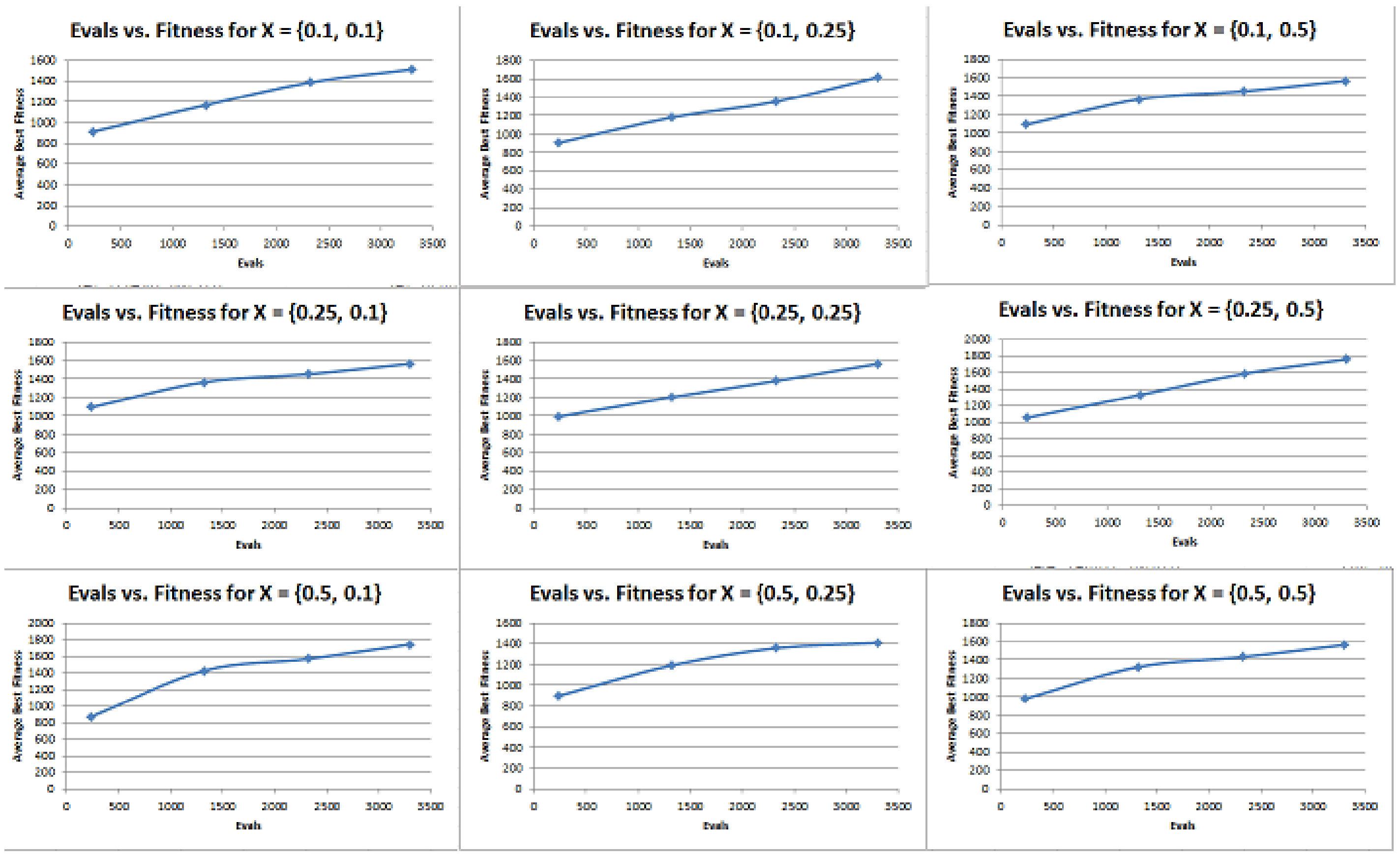

Figure 3: Fitness vs. Evaluation plots for all nine combinations of (Xp, Xg). Xp varies from {0.1, 0.25, 0.5} along the column while Xg varies along the row.

V. Discussion

A comparison of random search with GP shows that better Ms. Pacman agents are able to evolve with evolutionary search. Perhaps, the least satisfactory aspect of current work is the computational and time constraints on this project. There are several aspects that could have been improved. Experimental results from Ms. Pacman only evolution using GP showed significant promise. It'd be interesting to see how they fared against certain hand-coded behaviors of ghosts. Also, a comparison between exiting neural network evaluators and GP style expression trees would have provided some insight as to which representation is better. It may be that expression trees or function graphs involving logical, comparator and arithmetic functions are a better space to work in, since much of the reasoning behind location evaluation might be reducible to making distance comparisons between various objects in the maze and acting accordingly on the basis of those comparisons.

The best expression tree for Ms. Pac-Man in 2000 evals evolved as $-(\text{ManhattanDistPacmanGhost})^2$. With a random ghost controller, it makes a lot of sense to adopt ghost avoidance strategy.

With CoEA, 3000 evaluations are effectively four generations. Therefore, it is not surprising that CoEA was able to find the best Ms. Pacman controller with an average score of 1500, which is the same as random controller. To make it interesting, an experiment was conducted by running a CoEA for 50,000 evals with the same parameters as specified in Table IV to see if CoEA can really beat random search, while evolving interesting controllers for Ms. Pacman and ghost. Positively, CoEA was able to evolve a Pacman agent with 13000 average score. A breadth first enumeration of position evaluators for Ms. Pacman is shown below.

```
Sub(2) - Div(2) - Div(2) - Mul(2) - Add(2) - Sub(2) - Add(2) - Add(2) -
Sub(2) - Div(2) - Add(2) - Mul(2) - NearestManhattanDistPacmanGhost
- Sub(2) - DoubleConstant(0.46071576425778504) - Mul(2) - Rand(2) -
Div(2) - Mul(2) - Mul(2) - Sub(2) - Mul(2) -
NearestManhattanDistPacmanGhost - Rand(2) - Rand(2) - Mul(2) -
Rand(2) - DoubleConstant(0.9420192331935481) -
NearestManhattanDistPacmanPill - NearestManhattanDistPacmanGhost
- DoubleConstant(0.4363513394385691) -
DoubleConstant(0.5043048707275352) -
NearestManhattanDistPacmanPill - NearestManhattanDistPacmanPill -
NearestManhattanDistPacmanGhost -
NearestManhattanDistPacmanGhost - NearestManhattanDistPacmanPill
- DoubleConstant(0.026499780331128764) -
NearestManhattanDistPacmanGhost -
NearestManhattanDistPacmanGhost -
DoubleConstant(0.4518957840506160З) -
DoubleConstant(0.6915652952850276) -
DoubleConstant(0.9734072459722809) - Mul(2) - Div(2) -
DoubleConstant(0.46071576425778504) - Rand(2) -
NearestManhattanDistPacmanPill - NearestManhattanDistPacmanGhost
- NearestManhattanDistPacmanPill - NearestManhattanDistPacmanPill -
Sub(2) - DoubleConstant(0.4363513394385691) -
NearestManhattanDistPacmanGhost -
NearestManhattanDistPacmanGhost - Mul(2) -
NearestManhattanDistPacmanGhost - Rand(2) - Rand(2) -
DoubleConstant(0.6915652952850276) -
DoubleConstant(0.9734072459722809) - Div(2) - Mul(2) -
NearestManhattanDistPacmanPill - NearestManhattanDistPacmanPill -
NearestManhattanDistPacmanPill - NearestManhattanDistPacmanPill
```

Due to time constraints on this assignment, the resulting Ms. Pacman expression tree was not simplified. Ghosts however converged to 'ManhattanDistPacmanGhost', adopting Pac-Man chaser strategy. In the future, it'd be interesting to study controllers evolved by a CoEA with hall of fame and disengagement mechanisms in place.

VI. Conclusion

This paper investigated the use of expression trees as a possible representation for position evaluation as opposed to neural networks. In the first experiment, Ms. Pacman only evolution was conducted against random ghost controller using GP. In comparison to random search, GP performed exceptionally well proving that it has the capability to evolve better controllers than the baseline controller.

In the second experiment, competitive coevolution was used to evolve Ms. Pacman and ghost controllers simultaneously. It was determined that varying parent selection across both sub-populations had little or effect on convergence. This was however attributed low number of evaluations performed.

Finally, a CoEA run with 50,000 evals resulted in ghosts adopting a basic Pac-Man chaser strategy while Ms. Pacman evolved a complex controller scoring 13000 on average. Results in this report suggest co-evolutionary GP as a feasible approach towards the evolution of position evaluators for Ms. Pac-Man and Ghosts.

References

[1] Clark, D. (1997) "Deep Thoughts on Deep Blue," *IEEE Expert*, 12:4, p. 31.

[2] Tesauro, G. (1992) "Practical Issues in Temporal Difference Learning," *Machine Learning*, 8, pp. 257-277.

[3] K. Chellapilla and D. Fogel. Evolving an expert checkers playing program without using human expertise. IEEE Transactions on Evolutionary Computation, 5:422 – 428, (2001).

[4] J. Koza. Genetic Programming: on the programming of computers by means of natural selection. MIT Press, Cambridge, MA, (1992)

[5] M. Gallagher and A. Ryan. When will a genetic algorithm outperform hill climbing? In Proceedings of IEEE Congress on Evolutionary Computation, pages 2462 – 2469. 2003.

[6] J. S. D. Bonet and C. P. Stauffer. Learning to play Pac-Man using incremental reinforcement learning., 1999.

[7] http://csee.essex.ac.uk/staff/sml/pacman/kit/AgentVersusGhosts.html

[8] Simon M. Lucas, Evolving a Neural Network Location Evaluator to Play Ms. Pac-Man,IEEE Symposium on Computational Intelligence and Games (2005), pages: 203-210.