COMP SCI 5401 FS2017 - Iterated Prisoner's Dilemma: A Coevolutionary Genetic Programming Approach

Daniel Tauritz, Ph.D.

November 17, 2017

Synopsis

The goal of this assignment set is for you to become familiarized with (I) unambigously formulating complex problems in terms of optimization, (II) implementing an Evolutionary Algorithm (EA) of the Genetic Programming (GP) and Coevolutionary (CoEA) persuasions, (III) conducting scientific experiments involving EAs, (IV) statistically analyzing experimental results from stochastic algorithms, and (V) writing proper technical reports.

The problem that you will be solving is writing an EA to coevolve agents both cooperating and competing in the Iterated Prisoner's Dilemma (IPD) environment with a GP representation. This problem is representative of a large and very important class of problems which require the identification of system models such as controllers, programs, or equations. An example of the latter is symbolic regression which attempts to identify a system model based on a limited number of observations of the system's behavior; classic mathematical techniques for symbolic regression have certain inherent limitations which GP can overcome. Employing GP to evolve controllers for IPD agents is also a perfect illustration of how GP works while avoiding many of the complications of evolving full blown computer programs. This problem is also representative of a large and important class of problems where fitness is based on the interaction of individuals in one or more populations, so fitness is relative rather than absolute.

These are individual assignments and plagiarism will not be tolerated. You must write your code from scratch in one of the approved programming languages. You are free to use libraries/toolboxes/etc, except search/optimization specific ones such as Matlab's EA toolbox, C++/Java EA libraries, etc.

Problem statement

The Prisoner's Dilemma is a game where two criminals are captured, imprisoned, and forced to make a risk-reward decision [1]. They are each given the chance to defect and testify against their partner for the chance of being let free. They also know that their partner is being made the same offer. Their only goal is to reduce their own number of years in prison to a minimum (preferably zero). Choosing to not defect is referred to as choosing to cooperate. If prisoner A (self) chooses to defect while prisoner B (opponent) chooses to cooperate, then prisoner A will get out of all prison time while prisoner B will be convicted and held in prison for 5 years. If prisoner A chooses to defect but so does prisoner B, then both will have their testimonies rejected and be put away for 4 years. However, if both prisoners choose to cooperate, then they can only be convicted of a lesser charge and they will each get 2 years in jail. Table 1 summarizes this with the resulting payoff from the different combinations of choices, where payoff is equal to five minus the number of years in prison.

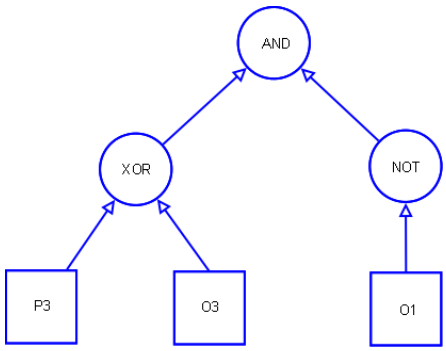
		Opponent	
		C	D
Prisoner	C	3	0
Prisoner	D	5	1

Table 1: The Prisoner's Dilemma payoff matrix for the prisoner agent (self).

The Iterated Prisoner's Dilemma (IPD) consists of a sequence of the same two individuals playing the Prisoner's Dilemma iteratively with some memory of the outcomes of the previous iterations. The general IPD optimization problem poses the question: what is the optimal strategy (controller) for an individual player (agent) in the IPD environment?

Example

The state of the environment consists of the last k outcomes. For this example assume that $k = 5$ and that the initial environment consists of the sequence [(0,1),(0,0),(1,1),(1,0),(1,0)] where the first element (0,1) specifies the outcome k iterations ago and the last element (1,0) describes the most recent outcome. Furthermore, 0 encodes defecting and 1 encodes cooperating; the element (0,1) specifies that the agent (always the first listed value) defected and its opponent cooperated. Now assume that the agent's controller is encoded by the following tree:



Circles indicate function nodes and boxes indicate terminal nodes. The left-most terminal indicates via the 'P' that the lookup should be for the prisoner agent and via the '3' that the lookup is for 3 outcomes ago. Per the sequence provided above, this is the value 1. The right-most terminal indicates via the 'O' that the lookup should be for the opponent agent and via the '1' that the lookup should be for the most recent outcome. Per the sequence provided above, this is the value 0. The middle terminal evaluates in the same fashion to 1. The XOR evaluates to 0. The NOR evaluates to 1. The AND finally evaluates to 0. Therefore, the agent decides to defect.

A solution file containing this example strategy in pre-order format would be as follows:

```
AND XOR P3 O3 NOT O1
```

General implementation requirements

Your programs need to by default take as input a configuration file *default.cfg* in which case it should run without user interaction and you must provide a command line override (this may be handy for testing on different configuration files).

The configuration file should at minimum specify:

- the value of l (the sequence length specifying the number of iterations to play),
- the value of k (the maximum length of the agent memory),
- the value of d (the maximum tree depth, where $d = 0$ would be a tree consisting of just a root which would be a single terminal node),
- an indicator specifying whether the random number generator should be initialized based on time or the seed for the random number generator (to allow your results to be reproduced),
- all algorithm parameters,
- the number of runs a single experiment consists of,
- the relative file path+name of the log file, and
- the relative file path+name of the solution file.

The log file should at minimum specify:

- k,
- d,
- the random number generator seed,
- the algorithm parameters, and
- an algorithm specific result log (specified in the assignment specific section).

For each run, a new uniform random initial state should be generated (but only initialize the random number generator once per experiment). The solution file needs to contain the best strategy found in pre-order format.

Strategies are encoded in GP style trees with the function set consisting of the logical operators AND, OR, NOT, and XOR and the terminal set consisting of either a P or an O followed by a number between 1 and k, where P indicates the prisoner agent (self) and O indicates the opponent agent, and the number following indicates the number of outcomes ago to lookup. The initialization for each run consists of randomly generating a sequence of length k consisting of binary pairs where the first pair contains the outcome of k outcomes ago and the last pair contains the most recent outcome, and the first element of each pair indicates the outcome of the prisoner agent (self) and the second element the outcome of the opponent agent. An outcome of 0 encodes defection and an outcome of 1 encodes cooperation. After each of the l iterations, the oldest pair of outcomes is dropped from the beginning of the outcomes sequence and the newest pair of outcomes is added at the end of the outcomes sequence.

The fitness of a strategy against a given opponent strategy is proportional to its average payoff after playing l rounds of IPD. Each round is scored according to Table 1 with scores being inversely proportional to the number of years the prisoner agent controlled by that strategy has spent in prison. For example, if an agent plays two rounds of IPD (so $l = 2$) and cooperates each time as does its opponent, then its fitness can be computed as its average sum of payoffs which is $(3+3)/2 = 6/2 = 3$; but if the agent plays a third round and again cooperates but his opponent defects, then its fitness using the same computation is its average sum of payoffs which is is $(3 + 3 + 0)/3 = 6/3 = 2$. Note that the fitness of its opponent after these three rounds would be $(3 + 3 + 5)/3 = 11/3 \approx 3.66$.

Note that your source code submissions need to include any necessary support files such as makefiles, project files, libraries, etc. so that they compile and execute "out of the box" on standard campus machines; non-compliance will result in an unacceptable high work load for the grader and therefore may be severely penalized.

GitLab requirements

For each assignment you will be given a new repository on [`https://git-classes.mst.edu`] **Please view your repository and the README.md** It may clear things up after reading this.

At the end of the assignment deadline, the code currently pushed to Master will be pulled for grading. You are required to include a run.sh that needs to be configured to compile and run with the command ./run.sh using a configuration provided by you. Specifically, in order to run your submission and grade your output, all the TA should have to do is execute:

```
./run.sh config
```

The TA should not have to worry about external dependencies. Any files created by your assignment must be created in the present working directory or subfolders in the present working directory.

Resubmissions, penalties, documents, and bonuses

You may commit and push to your repository at anytime. At submission time, your latest, pushed, commit to the master branch will be graded (if there is one). In order to ensure that the correct version of your code will be used for grading, after pushing your code, visit [`https://git-classes.mst.edu`] and verify that your files are present. If for any reason you submit late, then **please notify the TA when you have submitted.** If you do submit late, then your first late submission will be graded.

The penalty for late submission is a 5% deduction for the first 24 hour period and a 10% deduction for every additional 24 hour period. So 1 hour late and 23 hours late both result in a 5% deduction. 25 hours late results in a 15% deduction, etc. Not following submission guidelines can be penalized for up to 5%, which may be in addition to regular deduction due to not following the assignment guidelines.

Your code needs to compile/execute as submitted without syntax errors and without runtime errors. Grading will be based on what can be verified to work correctly.

Documents are required to be in PDF format; you are encouraged to employ LaTeX for typesetting.

Some assignments may offer bonus points for extra work, but note that the max grade for the average of all assignments is capped at 100%.

Assignment 2a: Random Search

Implement a random search to find within a configurable number of fitness evaluations the IPD prisoner agent strategy which incurs the least prison time versus a fixed tit-for-tat strategy (i.e., do whatever your opponent did last). You randomly create strategies by generating random trees of user-defined depth d and you determine the fitness of a strategy by playing it l times against the fixed strategy tit-for-tat.

The result log should be headed by the label "Result Log" and consist of empty-line separated blocks of rows where each block is header by a run label of the format "Run i" where i indicates the run of the experiment and where each row is tab delimited in the form <evals><tab><fitness> (not including the < and > symbols) with <evals> indicating the number of evals executed so far and <fitness> is the value of the fitness function at that number of evals. The first row has 1 as value for <evals>. Rows are only added if they improve on the best fitness value found so far. The solution file should consist of the best solution found.

The deliverables of this assignment are:

- Your source code (including any necessary support files such as makefiles, project files, etc.)
- One configuration file configured for 10,000 fitness evals, timer initialized random seed, and 30 runs, along with the corresponding log file and solution file

- A document headed by your name, S&T E-mail address, and the string "COMP SCI 5401 FS2017 Assignment 2a", containing the evals versus fitness plot corresponding to your log file which produced the best solution.

Edit your README.md file to explain anything you feel necessary. Submit all files via GitLab, by *pushing* your latest commit to the *master* branch. The due date for this assignment is Sunday, November 5, 2017.

Grading
The maximum number of points you can get is 75. The point distribution is as follows:

Algorithmic	40
Configuration files and parsing	5
Log file creation	5
Solution file creation	15
Good programming practices including code reliability and commenting	5
Evals versus fitness plot	5

Assignment 2b: Genetic Programming Search

Implement Genetic Programming to evolve within a configurable number of runs and a configurable number of fitness evaluations per run, the IPD prisoner agent strategy which incurs the least prison time versus a fixed tit-for-tat strategy (i.e., do whatever your opponent did last). The fitness of a strategy is computed a little bit differently than with random search, namely by first playing $2k$ IPD rounds before starting to count scores towards fitness (i.e., the fitness of a strategy against a given opponent strategy is proportional to its average payoff for the last $l - 2k$ rounds after playing l rounds of IPD). To ensure a sufficient number of rounds to average over, you need to enforce $l \geq 3k$.

You need at minimum to implement support for the following GP configurations, where operators with multiple options are comma separated:

Representation Tree as defined in the general implementation requirements section

Initialization Ramped half-and-half (see Section 6.4 in [2]) with $D_{max} = d$

Parent Selection Fitness Proportional Selection, Over-Selection (see Section 6.4 [2])

Recombination Sub-Tree Crossover

Mutation Sub-Tree Mutation

Survival Selection Truncation, k-Tournament Selection without replacement

Bloat Control Parsimony Pressure

Termination Number of evals, no change in fitness for n evals

Your configuration file should allow you to select which of these configurations to use as well as how many runs a single experiment consists of, l, k, and d. Optionally you can make it user-configurable whether you rerandomize the initial memory for each fitness evaluation (i.e., at the start of each sequence of l IPD rounds) or only randomize it once per run. If you really want to go all out, then you could even make it user-configurable to average fitness over multiple IPD sequences each with a rerandomized initial memory; note that this might make your run-time impractically large for the purpose of this assignment. Your configurable EA strategy parameters should include all those necessary to support your operators, such as:

- μ

- λ
- k for survival selection
- p for parsimony pressure penalty coefficient
- Number of evals till termination
- n for termination convergence criterion

The result log should be headed by the label "Result Log" and consist of empty-line separated blocks of rows where each block is headed by a run label of the format "Run i" where i indicates the run of the experiment and where each row is tab delimited in the form <evals><tab><average fitness><tab><best fitness> (not including the < and > symbols) with <evals> indicating the number of evals executed so far, <average fitness> is the average population fitness at that number of evals, and <best fitness> is the fitness of the best individual in the population at that number of evals (so local best, not global best!). The first row has $< \mu >$ as value for <evals>. Rows are added after each generation, so after each λ evaluations. The solution file should consist of the best solution (i.e., IPD strategy) found in any run.

The deliverables of this assignment are:

- Your source code (including any necessary support files such as makefiles, project files, etc.)
- A configuration file configured for 30 runs of 10,000 fitness evals with $k = 5$, $l = 30$, and $d = 10$, with the best EA strategy parameters you can find for your GP implementation
- The corresponding log file
- The corresponding solution file
- A document headed by your name, S&T E-mail address, and the string "COMP SCI 5401 FS2017 Assignment 2b", containing:
  - the corresponding plot of the global best fitness versus fitness evals; box plot is preferred,
  - your statistical analysis comparing the average final best random search result (note that in Assignment 2a you plotted the best run results as opposed to the average results) versus the average final best result you obtained in Assignment 2b (report said averages along with standard deviation, describe the test statistic you employed, and the appropriate analysis results for said test statistic).

Edit your README.md file to explain anything you feel necessary. Submit all files via GitLab, by *pushing* your latest commit to the *master* branch. The due date for this assignment is Sunday, November 19, 2017.

Grading
The maximum number of points you can get is 100. The point distribution is as follows:

Algorithmic	65
Configuration files and parsing	5
Logging and output/solution file creation	10
Good programming practices including code reliability and commenting	10
Document containing evals versus fitness plots	5
Statistical analysis	5

Assignment 2c: Coevolutionary Search

Implement single-population Coevolution and Genetic Programming to coevolve within a configurable number of runs and a configurable number of fitness evaluations per run, IPD prisoner agent strategies which incur the least prison time versus the other coevolved strategies (relative fitness) and test the fittest final population coevolved strategy (i.e., the locally best of the final generation over all runs) against a fixed tit-for-tat strategy (absolute fitness). The (composite coevolutionary) fitness of a strategy is computed by averaging the fitness of that strategy against a sampling of its opponents, as determined by the "coevolutionary fitness sampling percentage" strategy parameter where the minimum is one opponent and the maximum is $\mu + \lambda - 1$ opponents. Determining the fitness of a strategy against a particular opponent is accomplished by first playing $2k$ IPD rounds before starting to count scores towards fitness (i.e., the fitness of a strategy against a given opponent strategy is proportional to its average payoff for the last $l - 2k$ rounds after playing l rounds of IPDF); to ensure a sufficient number of rounds to average over, you need to enforce $l \geq 3k$. The definition of "one fitness evaluation" is determining the fitness of one strategy against one other strategy. So if sampling requires a strategy to be pitted against ten other strategies in order to determine its (composite coevolutionary) fitness, then that counts as ten fitness evaluations.

You need at minimum to implement support for the following configurations, where operators with multiple options are comma separated:

Representation Tree as defined in the general implementation requirements section

Initialization Ramped half-and-half (see Section 6.4 in [2]) with $D_{max} = d$

Parent Selection Fitness Proportional Selection, Over-Selection (see Section 6.4 in [2])

Recombination Sub-Tree Crossover

Mutation Sub-Tree Mutation

Survival Selection - Strategy Plus, Comma (comma a.k.a. generational)

Survival Selection - Technique Truncation, k-Tournament Selection without replacement

Bloat Control Parsimony Pressure

Termination Number of evals, no change in fitness for n evals

Your configuration file should allow you to select which of these configurations to use as well as how many runs a single experiment consists of, l, k, and d. Optionally you can make it user-configurable whether you rerandomize the initial memory for each fitness evaluation (i.e., at the start of each sequence of l IPD rounds) or only randomize it once per run. If you really want to go all out, then you could even make it user-configurable to average fitness over multiple IPD sequences each with a rerandomized initial memory; note that this might make your run-time impractically large for the purpose of this assignment. Your configurable EA strategy parameters should include all those necessary to support your operators, such as:

- μ
- λ
- k for survival selection
- p for parsimony pressure penalty coefficient
- Number of evals till termination
- n for termination convergence criterion

- Coevolutionary fitness sampling percentage (the minimum would be to determine fitness by comparing against a single opponent, the maximum would be to determine fitness by comparing against all opponents which is $\mu + \lambda - 1$

The result log should be headed by the label "Result Log" and consist of empty-line separated blocks of rows where each block is headed by a run label of the format "Run i" where i indicates the run of the experiment and where each row is tab delimited in the form <evals><tab><average composite fitness><tab><best composite fitness> (not including the < and > symbols) with <evals> indicating the number of evals executed so far, <average composite fitness> is the average composite population fitness at that number of evals, and <best composite fitness> is the composite fitness of the best individual in the population at that number of evals (so local best, not global best!). The first row has the number of evaluations after evaluating the initial population as value for <evals>. Rows are added after each generation. After the last generation row, an extra block should follow containing the absolute fitness of the fittest final population solution found against the tit-for-tat strategy. The extra block is headed by the label "Absolute Fitness" and contains a single row in the form <absolute fitness>. The solution file should consist of the best solution (i.e., IPD strategy) found in any run.

For the following experiments use $k = 5$, $l = 30$, $d = 10$, and a minimum of 10,000 fitness evals; more is allowed, as long as all the experiments use the same number of fitness evals. Informally experiment with the sensitivity of the final local best to the EA strategy parameters to determine which parameter seems to make the most difference on both the relative fitness and the absolute fitness. Then formally experiment with the sensitivity of the final local best to that parameter by at minimum trying three different values for it and collecting both relative fitness and absolute fitness statistics for 30 runs. Use an appropriate statistical test (e.g., t-test or anova) to determine with $\alpha = 0.05$ which combinations are statistically better in terms of final local best. Make three plots, one for each combination, with each plot showing evals versus both local average fitness averaged over the 30 runs and global best fitness averaged over the 30 runs.

The deliverables of this assignment are:

- Your source code (including any necessary support files such as makefiles, project files, etc.)
- The three configuration files and corresponding three log files, and three solution files
- a document headed by your name, S&T E-mail address, and the string "COMP SCI 5401 FS2017 Assignment 2c", containing:
  - a methodology section describing your CoEA design in sufficient detail to allow someone else to implement a functionally equivalent CoEA; make sure to explain your CoEA design decisions.
  - an experimental setup section describing your experimental setup in sufficient detail to allow exact duplication of your experiments (i.e., producing the exact same results within statistical margins); make sure to justify your choice of CoEA strategy parameters.
  - a results section containing three plots corresponding to the earlier listed deliverables, box plots are preferred, a table containing averages and standard deviations of the three combinations for both relative and absolute fitness, and your statistical results of the three combinations for both relative and absolute fitness.
  - a discussion section where you discuss your experimental and statistical results, providing valuable insights such as conjectures you induce from your results; your choice of what to report on and how you go about rationalizing it is your subjective interpretation.
  - a conclusion section where you state your most important findings and insights.

Edit your README.md file to explain anything you feel necessary. Submit all files via GitLab, by *pushing* your latest commit to the *master* branch. The due date for this assignment is Sunday, December 3, 2017.

Grading
The maximum number of points you can get is 125. The point distribution is as follows:

Algorithmic	65
Configuration files and parsing	5
Log file creation	5
Solution file creation	5
Good programming practices including code reliability and commenting	5
Methodology section	10
Experimental setup section	5
Results section	10
Discussion section	10
Conclusion section	5

Bonus Options
All bonus investigations need to be documented in separate sections of the required document marked as "Bonus x" where x corresponds to the below Bonus number. You also need to indicate in your source files any code which pertains to a particular bonus and additionally describe it in your README.md file. Basically, you need to make it as easy as possible for the TA to see with a glance what part of your submission pertains to a particular bonus, and which doesn't. If you decide to do both bonus options, then you should discuss your investigations of how well they work in an integrated manner.

Bonus 1 You can earn up to 15 bonus points for adding a monitor to detect whether cycling has occurred. You need to describe the exact mechanism of your monitor and investigate how well it works.

Bonus 2 You can earn up to 15 bonus points for adding a technique to deter cycling (e.g., a hall of fame). You need to describe the exact working of your technique and investigate how well it works.

References

[1] Melanie Mitchell *An Introduction to Genetic Algorithms.* The MIT Press, Cambridge, Massachusetts, 1997, ISBN 0-262-13316-4.

[2] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing.* Second Edition, Springer, 2015, ISBN 978-3-662-44873-1.