RESEARCH SCHOOL OF COMPUTER SCIENCE

COLLEGE OF ENGINEERING AND COMPUTER SCIENCE

An Implementation of MC-AIXI-CTW

Jarrah Bloomfield[*] Luke English[†] Andrew Haigh[‡] Joshua Nelson[§] Anthony Voutas[¶]

COMP4620 - ADVANCED ARTIFICIAL INTELLIGENCE
ASSIGNMENT 2

vspace1.4cm

Bottom of the page October 31, 2012

u4669875[*] u4667010[†] u4667844[‡] u4850020[§] u4519169[¶]

Chapter 1

Description of the MC-AIXI-CTW implementation

The AIXI agent is a formal, mathematical agent which represents a solution to the general reinforcement learning problem. Principally, AIXI consists of an expectimax search over a Bayesian mixture of Turing machines in order to choose optimal actions by predicting future observations and rewards based on past experience. Given infinite computational resources, AIXI represents the optimal reinforcement agent: maximising future expected reward in any unknown environment.

In the far more limited world of what is tractable, we require an approximation to AIXI. Here, we approximate AIXI via a combination of UCT search (Monte-Carlo Tree Search with the Upper Confidence Bound) (Kocsis and Szepesvári, 2006) and Context Tree Weighting (Willems et al., 1995), yielding MC-AIXI-CTW.

Chapter 2

User Manual

2.1 Arguments

The agent can be compile with the `make` command. The agent then can then be run using

```
./main <environment> <logfile>
```

`<environment>` is a compulsory argument, which specifies the environment configuration file the agent is to use. In this implementation, it is one of the following

- `coinflip.conf`: Biased coin flip enviroment
- `grid.conf`: Gridworld environment
- `kuhnpoker.conf`: Kuhn poker environment
- `pacman.conf`: Pacman environment
- `rps.conf`: Biased rock paper scissors environment
- `tiger.conf`: "Tiger" Environment
- `composite.conf`: A combination of the above environments

`<logfile>` is an optional argument, which specifies the name of a log file to output results to.

2.2 Configuration files

`.conf` files are *configuration files*, specifying which environment is to be used, and relevant parameters for each environment. Each configuration file has the following parameters

- `environment`: The name of the environment to use. One of {4x4-grid, kuhn-poker, tiger, biased-rock-paper-scissor, pacman, composite}.
- `exploration`: The rate at which the agent explores, by making random decisions.
- `explore-decay` : The rate at the exploration rate decreases

In addition to this, some configurations have parameters that are specific to their environments.

- Coinflip
  - `coin-flip-p`: The probability of a flipping heads ($0 \leq$ `coin-flip-p` ≤ 1).
- Kuhn poker

  - `gamma`: A constant that determines the environment's Nash equilibrium strategy. ($0 \leq$ `gamma` ≤ 1)

- Pacman
  - `mapfile`: The location of the map file for the pacman board.

- tiger.conf
  - `left-door-p`: The probability that the gold is behind the left door
  - `listen-p`: The probability that a listening observation is correct.

- composite.conf
  - `environmentN`: Specifies the N^{th} environment, where $0 \leq N \leq 10$. The value of this parameter is an integer ≤ 10, and indicates which environment environmentN represents.
  - `startN`: Specifies the time step that at which the N^{th} environment starts, where $0 \leq N \leq 10$.
  - Paremeters required for the environemnts $1..N$ specified in `environment.cpp`.

Chapter 3

MC-AIXI-CTW Implementation

3.1 Design Choices

3.2 Context Tree Weighting

The algorithm for implementing context-tree weighting (CTW) mainly consists of the methods and objects given in the file `predict.cpp`, along with its corresponding `.hpp` file. We run through these from the beginning of the file to its end.

3.2.1 Objects

- `CTNode`

  This object represents a node in the context tree; and so it needs to store both the Laplacian estimate of the probability, along with the weight given to it by the actual CTW algorithm. For mathematical stability, we store these floating-point numbers as logarithms, so we can just add them instead of multiplying them. In addition, it has a 2-array of pointers to its left and right child, and it stores the counts for these nodes in the context of the history (which is also required for the CTW algorithm).

- `ContextTree`

  Here we have the entire context tree represented as an object, which is a glorified pointer to the root node of the tree (as `CTNode`s store their own children). In addition to storing the root, `ContextTree` also stores a double-ended queue of symbols (binary digits) which represents the current history of the tree, and an unsigned integer which represents the maximum depth of the tree (so that we don't add nodes past the depth, and take up more memory than we want).

3.2.2 Methods

`CTNode`

- `logProbWeighted, logProbEstimated, visits, child`
- `update`
- `size`
- `logKTMul`
- `revert`
- `prettyPrintNode`

`ContextTree`

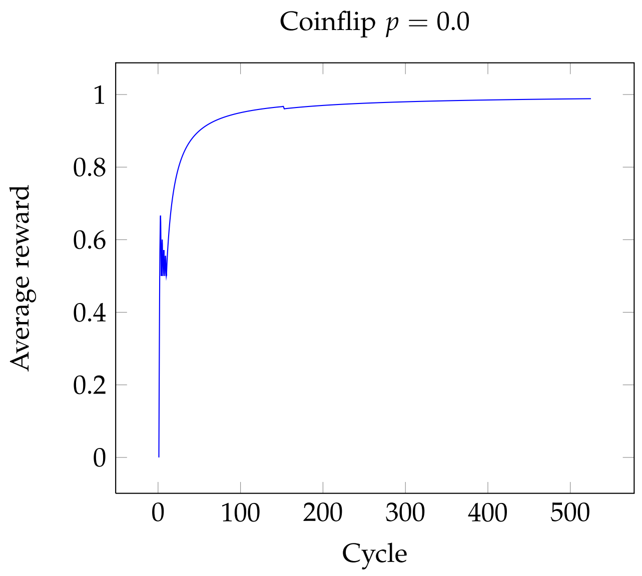
- `clear`
- `update`
- `updateHistory`
- `revert`
- `revertHistory`
- `predict`
- `genRandomSymbols`
- `genRandomSymbolsAndUpdate`
- `logBlockProbability`
- `nthHistorySymbol`
- `depth, historySize, size`
- `predictNext`
- `prettyPrint, prettyPrintNode, printHistory`

3.3 Upper Confidence Tree

3.4 Revert Function

Chapter 4

Experimentation



4.1 Learning

4.1.1 Sequence prediction

The CTW tree is the primarmy prediction mechanism in the MC-AIXI-CTW model. The CTW implementation of MC-AIXI-CTW was tested in isolation from the rest of the agent on a range of deterministic and non-deterministic sequence.

4.2 Extension

Bibliography

L. Kocsis and C. Szepesvári. Bandit based monte-carlo planning. *Machine Learning: ECML 2006*, pages 282–293, 2006.

F.M.J. Willems, Y.M. Shtarkov, and T.J. Tjalkens. The context-tree weighting method: Basic properties. *Information Theory, IEEE Transactions on*, 41(3):653–664, 1995.