CFS-C/FSW1: An Implementation of the CFS-C Classifier System in a Task-Domain that Involves Learning to Traverse a Finite State World

Rick L. Riolo

January 1988

Logic of Computers Group
Division of Computer Science and Engineering
University of Michigan

†This work was supported in part by National Science Foundation Grants DCR 83-05830 and IRI 86-10225, and by a University of Michigan Presidential Initiaive Fund Grant to the BACH research group.

Contents

1 Introduction

The FSW1 program enables a user to set up a wide variety of tasks for the CFS-C classifier system [Riolo, 1988a], each modeled as a finite Markov process in which a *payoff* is associated with some states. The classifier system's input interface provides a message that indicates the current state of the Markov process. The classifier's output interface provides the system with a way to alter the transition probabilities of the process, so that the system can control (in part) the path taken through the finite state world. When the classifier system visits states with non-zero payoff, that payoff is given to the system as a "reward". Thus the task for the CFS-C classifier system in the FSW1 domain is to learn to emit the appropriate signals at each step so that the Marlov process will visit states with higher payoff values as often as possible.

This document describes the FSW1 task domain, an example of the domain-dependent part of a classifier system designed to be linked to the CFS-C classifier system[Riolo, 1988a]. For general information about the CFS-C package of subroutines which implements the domain-independent parts of a learning classifier system, see the cited documentation. Th CFS-C documention also describes how to design your own task domains for the CFS-C system, and it describes all the commands and features provided by that system. For another example domain, see [Riolo, 1988b].

Note well: While I do attempt to keep this document up-to-date, that is note always possible. So if you have any questions or problems, please look at the FSW1 source files for the definitive answers about the version you have. Also please feel free to contact me with questions, comments, suggestions and, of course, bug reports. See the README.TXT file on the distribution disk for information on how to contact me.

2 Defining a FSW1 Environment

Formally, a particular FSW1 task domain is fully defined by specifying:

- A set of n states W_i, $i = 0, \ldots, n-1$, each with an associated payoff $\mu(W_i) \in \Re$; one state also is designated the start state.

- A set of probability transition matrices, $P(r)$, where each entry $p_{ij}(r)$ in $P(r)$ gives the probability of going to state W_j, given that the system is in state W_i and that the classifier system has emitted r as its output value ($r = 0 \ldots 15$).

For example, consider the simple three state world shown in Figure 1. (In this diagram, states are shown as boxes and arrows designate non-zero probability transitions.) W_0 is the start state. The payoff for state W_1 is 100; the payoff for the other states is 0. When the system is in state W_0, if $r = 1$ the probability of going from state W_0 to W_1 is 1.0; if $r = 2$ the probability of going from state W_0 to W_2 is 1.0. For other values of r, the probability of going from state W_0 to state W_1 or W_2 is 0. The probability of going from either W_1 or W_2 to W_0 is 1.0, no matter what the value of r. Thus if the classifier system is to maximize its payoff in this world, it must learn to set $r = 1$ whenever it is in state W_0.

Thus each particular FSW1 environment consists of a set of states and transition paths from each state to some other states. There is a designated start state, and some states

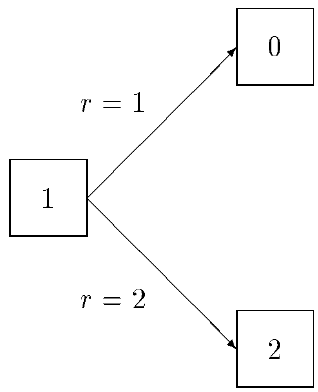

Figure 1: A simple FSW1 finite state world.

have non-zero payoff values associated with them. In the FSW1 implementation, each state is stored as an entry in an array of StateNd structures. Each contains information about the state, including:

1. Its state number (id).

2. The payoff associated with the state.

3. A count of the number of times the state has been visited.

4. An attribute list (binary string) that is used to generate a detector message when the state is the current state. A state will have an attribute string or it will have the NoAttr flag set TRUE, in which case no detector message is generated when the state is the current state.

5. An optional list of variable loci in the attribute list, with an associated probability that the detector message will contain a 1 at that locus. The specification of variable loci overrules the fixed values specified in the state attribute string.

6. A list of transitions from the state to other states. There is a list of transitions for each valid effector value r; these specifty the matrix $P(r)$ described earlier. Each transition list is given in terms of a list of $(s\prime, p)$ tuples, where $s\prime$ is a possible next state and p is the probability that $s\prime$ actually will be the next state. That is, the tuple $(s\prime, p)$ specifies the probability value $p = p_{ij}(r)$ in $P(r)$.

7. An optional "best next state,", and a count of the number of times the system went to that state, and a count of the number of times the system went to a state with a negative payoff.

A FSW1 environment is loaded from a file. (By default when the FSW1 system is started, it tries to load an environment from the file INIT.ENV. If no such file exists, a warning message is displayed and the user must use the CFS-C LOADENV command to load an environment.)

The following shows the format for a simple environment:

```
; The number of states followed by the initial state.
4, 0
; The state numbers (ids), visit counts, attribute strings,
; and variable loci (e.g., locus 4 will have 90% 1 and 10% 0).
0, 0, 0000 0000 0000 0000
1, 0, 0000 0000 0000 0001, 4,0.9, 5,0.1
2, 0, 0000 0000 0000 0010, 12,0.5
3, 0
ENDSTATES
; Now the payoff function, consisting of  s,u/ pairs separated
; by slashes for all states s with non-zero m  payoffs,
; followed by the ENDSTATES constant.
1,10/ 2,5
ENDPAYOFFS
; Now the transition function tuples (s,r,s1,p1,s2,p2,...).
; The s is the current state, the other s1, s2, etc.,
; are possible next states.
; The r's are effector values from the set (0,1,...EFFVALSZ).
; The first value entered is the default "do nothing" setting.
; It is used when no effector message are generated, as well
; as when the messages vote to do nothing (i.e. for r = 0).
; For each s,r combination:
; specify a list of destination states and probabilities for
; each being the path taken. If the probabilities don't add
; up to 1.0, you will get a warning on loading.
; NOTE: A line may be of the form "2,r, 3" ,
; which means for any EffSet,
; transition is from state 0 to 3 (MaxRInit) at random.
;
0,0, 0,.5, 1,.5
0,1, 1,.5, 2,.5
0,2, 1,.3333, 2,.3333, 3,.3334
0,3, 1,.3333, 2,.3333, 3,.3334
;
1,0, 1,.5, 2,.5
1,1, 2,.5, 3,.5
1,2, 2,.3333, 3,.3333, 0,.3334
1,3, 2,.3333, 3,.3333, 0,.3334
;
```

```
2,0, 1,.5, 2,.5
2,1, 3,.5, 0,.5
2,2, 3,.3333, 0,.3333, 1,.3334
2,3, 3,.3333, 0,.3333, 1,.3334
;
3,0, 3,.5, 0,.5
3,1, 0,.5, 1,.5
3,2, 0,.3333, 1,.3333, 2,.3334
3,3, 0,.3333, 1,.3333, 2,.3334
ENDTRANS
; The effectors. For Effector1, the right most 4 pairs are
; used to pick a transition path (see StoreSup for details).
 10## #### ## ## ## ##, Effector1, 0, 0
~10## #### ## ## ## ##, Default, 0, 0
END-EFFS
```

Note that the lines beginning with semicolons are ignored, and that lines should not be longer that 100 charcaters. Check the file FSW1.H for the limits on the number of states, transitions, and so on, that can be loaded. Depending on the computer/compiler you are using, you may be able to make the limits much larger than those set in FSW1.H on the distribution disk.

3 Detector Messages

Typically one detector message is added to the message list at each step; the message added is generated from the attribute string of the current state, with modifications for any variable loci for that state. (Following the usual convention, the left loci are always "00" to indicate the message is from the detectors.) If the current state has no attribute string, no detector message is added.

4 Effectors

The system's effector interface works as follows: All new messages that start with "10" in the leftmost loci are interpreted as effector messages. Each message is taken to be a "vote" for (or "support") setting the effector output value to one of a *set* of "r" values. The message intensity (i.e., the amount bid to produced it) is the amount of support for each r in the specified set. The actual r value is chosen probabilistically, with the probability that r is set to r' proportional to the total support for r' from all effector messages. For example, one message may support setting r to 1, 2, or 4, with a bid of 100, while another message may support setting r to 4 or 7 with a bid of 200. The total support is 700, and the total for $r = 4$ is 300, for $r = 7$ it is 200, while for $r = 1$ and 2 it is 100. Thus r is most likely to be set to 4.

Once an r value is chosen, that value is used to select a transition probability matrix $P(r)$, which in turn is used to select that next state of the system. Also, all effector messages that are not consistent with the r chosen (i.e., effector messages that specify sets that do *not* contain r) are deleted from the message list. (See the CFS-C documentation for a further discussion of *effector conflict resolution mechanisms* in classifier systems.)

How is an effector message mapped into a set of r values to be supported? First I will describe the default method, after which I will decribe ways the user can change the default behavior.

By default, the rightmost 8 bits of an effector message are decoded as follows to determine which r values are supported: Each pair of the 8 loci is mapped into one string value as follows:

$$
\begin{aligned}
00 &\rightarrow 0 \\
01 &\rightarrow 1 \\
10 &\rightarrow \# \\
11 &\rightarrow \#
\end{aligned}
$$

Thus the rightmost 8 bits are mapped into a string of length 4 over the $\{0, 1, \#\}$ alphabet, called the *effector value attribute string.* That 4 loci string is then interpreted as designating a *set* of r values supported by the effector message, treating the # as a "wildcard" in the usual way. For example, the effector message:

1000000000100100

is mapped into the effector attribute string: 0#10 which specifies the set of r values $\{3, 6\}$.

The user may also modify the way effector messages are mapped into set of r values by issuing CFS-C "environment specific" commands (the ECMD commands). In particular:

- The number of loci mapped into r values may be changed from the default of 8 to 2, 4, or 6 by issuing the command:

  ECMD EAS n

  where "n" may be 2, 4, 6 or 8. In each case the rightmost "n" loci of effector messages will be used to determine the set of supported r values. (Thus change the number of loci to be used changes the range of possible r values.)

- Loci other than the rightmost loci may be specified as designating the set of r values by using the command:

  ECMD EL $l_1, l_2, \vdots$

  where each l_i specifies a loci to use (counting from the left of the effector messate as 0). There must be 2, 4, 6, or 8 loci specified, and they should not include 0 or 1. The loci are read from left to right into a string that is then mapped into the effector attribute string.

- By default, if the effector value r is set to a value not listed for the current state, the first r value listed for the current state is chosen. The command:

  ECMD HTR 1

causes the system to pick an r value at random from the values listed for a state when an invalid (not listed) r is chosen. The command ECMD HTR 0 returns the system to the default behavior.

- By default, if there are no effector messages produced during a cycle step, the first r value listed for the current state is chosen. The command:

  ECMD DFTR 1

  causes the system to pick an r value at random from the values listed for a state when no effector message is produced. The command ECMD DFTR 0 returns the system to the default behavior.

5 Payoff

After the effector messages have been used to choose an effector value r, that value (along with the current state) is used to choose the next current state.

If that new current state has a non-zero payoff associated with it, that value is added to some or all of the classifiers active (i.e., which posted messages) at that time step. By default, the *full* payoff is added to *all* active classifiers. The CFS-C system provides a number of run-time settable parameters that can be used to control the allocation of payoff (reward) from the enviroment, including:

- *ShareRew.* If ShareRew is 0 (the default), the full reward is allocated to all eligible classifiers. If ShareRew is 1, the payoff is divided equally amoung the eligible classifiers.

- *FrPayNEC.* If FrPayNEC is 1 (the default), all active classifiers (those that *posted* messages not eliminated by the effector resolution mechanism) have the payoff (or payoff share) added to their strengths. If $0 \leq FrPayNEC < 1$, classifiers that produced effector messages still have the full reward (or share) added to their strengths, but other non-effector message produces have $FrPayNEC * Reward$ added to their strengths.

See the CFS-C documentation for more information about these and other CFS-C parameters.

6 Using the FSW1/CFS-C System

To produce a runable version of the FSW1 system, first compile all the files in the CFS-C system (14 currently), then compile the files FSW1.C and FSW1A.C, and finally link those two files with the 14 domain-independent files of CFS-C.

To run the system, set up an INIT.ENV file and then enter:

fsw1 path

where **fsw1** is the name you gave to the linked system, and **path** is a path descriptor to the location of the INIT.CFS file, a file that contains a list of all CFS-C run-time settable variables (see the file INTRO.TXT on the distribution disk for more information about that file). You can then enter various CFS-C or FSW1 (ECMD) commands to load classifiers (or generate random ones), display classifiers, messages, variables, etc., set variables to new values, classify for a number of steps, stop, and so on. You can also load an environment from files other than INIT.ENV using the LOADENV command. For details on all the commands, variables, and features available, see the CFS-C documentation and the TXT files on the distribution disk.

Also, there are some other commands specific to the FSW1 domain which can be executed using the CFS-C ECMD:

1. By default, when no effector message is generated the first r value is used as a "default" behavior, and the MadeMstk ("made mistake") flag is set TRUE. The command:

   ECMD DFTM 1

   tells the system to *not* set the MadeMstk flag when a default transition is used. (The MadeMstk flag acts a trigger for some of the discovery learning opertors of CFS-C.)

2. The command:

   ECMD CS s,E

   can be issued to set currnet state to "s", where "s" is a state id. the optional ",E" directs the system to also empty the message list.

3. The command:

   ECMD NBS s,b

   can be issued to designate state "b" as the "next best state" from state "s". When a state has a next best state, a count of the number of times the next best state is actually the next state chosen, which can be display with the DISPLAY ENV command.

The file FSW1-HLP.TXT describes these and other ECMD commands available in the CFS-C/FSW1 system.

Finally, note that when displaying classifiers (using the CFS-C *display cl* command or the auto-display facility) the FSW1 domain provides two *interpreted* formats for displaying classifiers, with numbers 21 and 22. If you display classifiers using format 21 (e.g, using *display cl,21*), each classifier will be shown in a format like the following:

id> stag: string [states]; stag: string [states] =actt=> stag: string [r] $\{s\}$ use

where:

id Is the classifier id number.

stag Is the "source-tag", i.e., the leftmost 2 bits interpreted as follows: 00 $\rightarrow D$ (macthes detector message), 10 $\rightarrow G$ (matches an effector message), 01 $\rightarrow X$ and 11 $\rightarrow Y$ (matches "internal," non-interface messages).

string For format 22 only, the uninterpreted form of the condition or action (not including the source tag).

states A list of any states matched by the condition.

r A list of any effector values supported, i.e., the rightmost bits interpreted as described earlier.

actt The action type, eg, PASS, AND, etc. (If PASS, its not writen).

s The classifier's current strength, bidratio and support, in braces.

use How rule was used in the most recent step: *b* means it bid, *w* means it won, and *p* means it posted messages (after effector conflict resolution.)

Note that if the detector message bits for some states have values that are determined at random, as described in the example INIT.ENV file in section 2, then the conditions that claim to match those states may only match them some of the time. For example, if a locus is assigned a value of 1 30% of the time and 0 the rest, and a condition has a 0 at that locus, then that condition will match the state only some of the time. For more details, see the source codes in FSW1A.C, or just try some examples and compare the interpreted to uninterpreted forms of the classifiers.

A Bibliography

References

[Riolo, 1988a] Riolo, Rick L. "CFS-C: A Package if Domain Independent Subroutines for Implementing Classifier Systems in Arbitrary, User-Defined Environments." Logic of Computers Group, Division of Computer Science and Engineering, University of Michigan, Ann Arbor, 1988.

[Riolo, 1988b] Riolo, Rick L. "LETSEQ1: An Implementation of the CFS-C Classifier System in a Task-Domain that Involves Learning to Predict Letter Sequences." Logic of Computers Group, Division of Computer Science and Engineering, University of Michigan, Ann Arbor, 1988.