BayesACT Version 2: Technical User Manual

Jesse Hoey
David R. Cheriton School of Computer Science
University of Waterloo
Waterloo, Ontario, N2L3G1
jhoey@bayesact.ca

Abstract

This paper gives is meant to serve as a technical guide to *BayesAct* (version 2, currently 2.3.8). Software is available for download at `bayesact.ca`, along with all relevant publications and links. Email the owner for access to the code at the address under the title. An online version of *BayesAct* is accessible through this link: `https://cs.uwaterloo.ca/~jhoey/research/bayesact/code/index.html`. A longer version of this paper exists that includes basic background material on POMDPs, planning, and affect control theory, also available by request to the author (email above). Section 1 introduces the software and some basic simulation examples are described in Section 2.

Working paper. DO NOT CITE. If you need to cite this paper, contact the author directly.

1 *BayesAct* v2 software

This section gives a practical overview of the new BayesACT code. This information is also found in the README. The code is available at `bayesact.ca`. Email the owner for access at the address shown above. It comes with the following copyright disclaimer:

```
#  Use for research purposes only.                                    #
#  Do not re-distribute without written permission from the author    #
#  Any commerical uses strictly forbidden.                            #
#  This program is distributed WITHOUT ANY WARRANTY;                  #
#  without even the implied warranty of                               #
#  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.               #
```

1.1 Differences between *BayesAct* v2 and v1

BayesACT Version 2 makes some substantial changes over BayesACT Version 1. It is written in C, not Python, so its now faster. A python wrapper is provided for convenience. Other major changes are as follows.

- The "somatic transform" is used for all connotative-denotative links. This was proposed in Neil MacKinnon and Jesse Hoey paper at the Identity conference. Its a major advance over the original BayesACT as provides a conceptual and mathematically well defined linkage between connotative (sentiment/affective) meanings and denotative (cognitive) states. See (Hoey et al., 2021; MacKinnon and Hoey, 2021) for more details on the somatic transform.

- The denotative state (labels) are explicitly taken into account, so each sample is weighted by its distance from the sentiment distribution of its denotative label (using a somatic transform).

- Re-identification is done on a **per sample** basis, so each agent maintains a running distribution over denotative labels. These are added as needed when the sample weights get too small, so the set of labels describing an agent grows and shrinks with the simulation (using a somatic transform). When sample weights get too small, agents try to re-identify first *client* identity, then (optionally) *agent* identity, then both simultaneously.

- Full variance information can be taken into account for dictionaries. The recent 2015 dataset is included with the zip file in which the full measured covariances are computed for each identity label.

- All identities are specified denotatively for a simulation and a simulation is specified using a simple language - see e.g.`simtest-base.txt` described further below.

- all parameters can be modified in this simple language

- Multiple agents can interact (see below Section 1.7).

- The planning engine is now a belief-state forward search. Actions and observations are sampled and a belief state is propagated to each node. The idea is to do a fairly shallow, precise, targetted search. The search goes for precision in a targetted (by the affect control principle) area rather than depth in an expected (according to the utility function) target area. The planning algorithm naturally trades off between rational decision theoretic (individual utility maximization) and affective alignment. **This is work in progress, and has not been fully tested in version 2.3.8, so use with caution.**

1.2 Installation

Requires the following:

- `gcc` - on Mac you can download the `XCode` package. On Windows, I have no clue - ask `@BillGates`. You will probably need some $. Linux, you probably will never read this, but if you do, visit `gnu.org`.
- `GSL` - get it from `www.gnu.org/software/gsl`
- `Python` version 3 for the python wrapper (optional)

To install, open a terminal, find the file and

```
>unzip bayesact-master.zip
>cd bayesact-master/source
>make
```

You may need to edit the `Makefile` first. If you are on linux, then `Makefile.linux` may work better for you. The default is `Makefile.macos`.

To use:

```
> ./bayesactsim -h
```

displays a usage message

A basic simulation can be run with:

```
> ./bayesactsim ../examples/simtest-base.txt
```

where `simtest-base.txt` is the simulation specification file to read from (see the files themselves and Section 1.7 below for more information). You may need to adjust your PATH or otherwise change the input file `simgtest-base.txt` so it points to the dictionaries (in the directory `data`).

The code comes with five applications, specified as a set of funtions in the files `defaultAPP.c`, `institutionAPP.c` `bayesactAPP.c`, and `pdAPP.c`. The first, `defaultAPP.c` models basic ACT-like simulations with no additional reward function and no state $\mathbf{X_x}$, while the second (`institutionAPP.c`) adds the ability to specify institutions. The third, (`bayesactAPP.c`) models a full bayesACT agent that maintains identity and takes institutions into account. Finally, (`pdAPP.c`) models a social dilemma and includes elements of state $\mathbf{X_x}$ and reward. To swap one out for the other you need to

- change the `Makefile` so that the line `APP=...` reads the appropriate thing (`pdAPP` or `defaultAPP`), and

- change the `#include ...` to read the same thing (`pdAPP.h`, `bayesactAPP.h`, or `defaultAPP.h`).

On the to-do list is to normalize this so the elements of additional state and all the dynamics are specified in a second text-based file, so the code does not have to be re-compiled to switch applications. For now, interested users can define their own applications by implementing the same functions as in those files. By default it uses the most general `bayesactAPP.c`, which sould be sufficient for most purposes.

See the `README` for more information on a number of topics. In the following we go over the basic elements.

1.3 Main Code Structure

The basic structures are found in `fvarsample.h`, the major functions in `fvarsample.c`. The main program is in `bayesactsim.c`. There are the following major components, also shown schematically in Figure 1.

1. there are a number of enums and other constants defined - see `fvarsample.h` for details. Of note:
   - `SSD` is the dimensionality of the emotion space (3 by default: EPA)
   - `NGT` is the number of grammar terms - in case we want to add settings in later
2. agents are agentState objects which have a name, filenames for transient impression and emotion dynamics parameters, parameters α, γ, δ a pointers to dictionaries for identities, behaviours and modifiers
3. dictionaries represent sentiment dictionaries. These are not replicated, so more than one agent can point to the same dictionary. They are used by the somaticTransform to map between connotative and denotative state. Dictionaries can be of four types (`enum dict_type`)
   (a) `MEAN` - a label followed by SSD double
   (b) `SD` - a label followed by SSD doubles (the mean), followed by SSD doubles (the std. devs. on the mean)
   (c) `COV` - a label followed by SSD doubles (the mean), followed by SSD^2 doubles (the covariances on the mean)
   (d) `SAMPLE` - (raw set of samples - **not implemented yet**)
   (e) `GAUSS_MIX` - (a label followed a number M - number of mixtures, followed by M sets of weights + COV or SD parameters - **not implemented yet**)

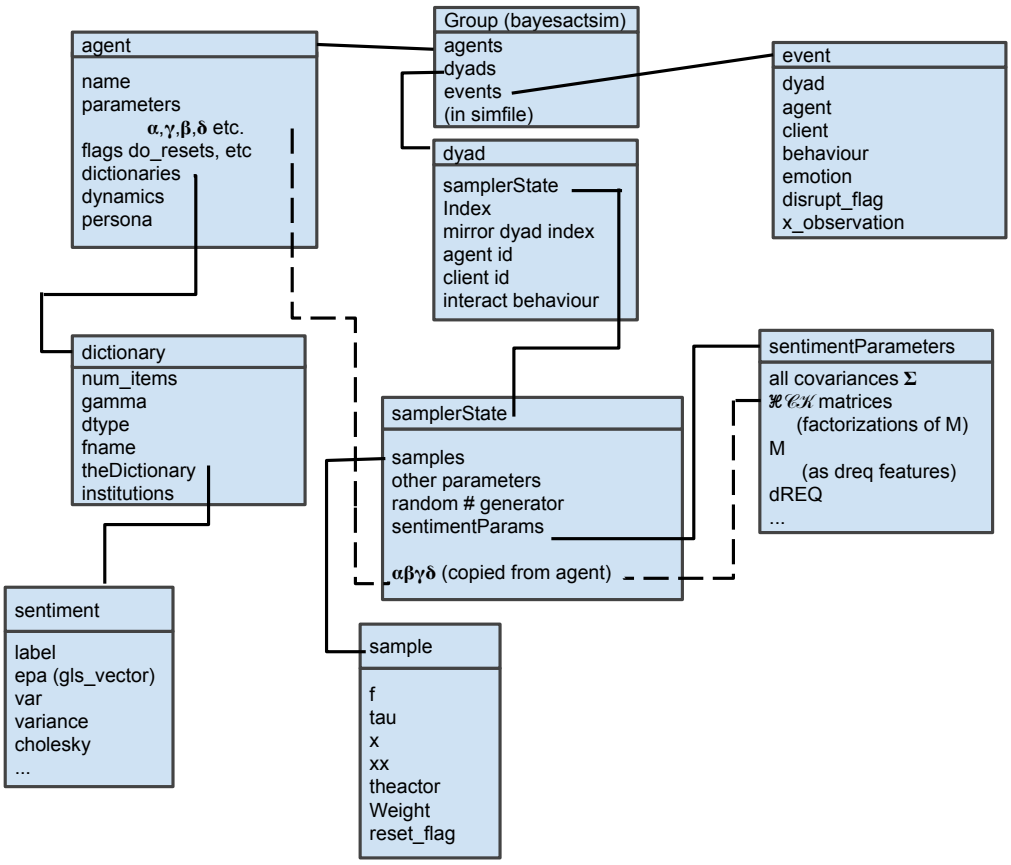

Figure 1: This schematic shows how BayesACT 2.3.8 works. The "Group" is really the bayesactsim simulator file at this point (but could be encapsulated). It consists of a set of agents, a set of dyads, and a set of events. These are specified in a simulation file (see below). The agent has a set of parameters (alpha, beta, gamma), a set of dictionaries, an emotional dynamics matrix (impression formation equations) and a persona (a set of weighted identities). A dyad has a samplerState that contains all it needs to run a simulation, including copies of the key parameters from the primary agent for the parent dyad. It also contains the parameters suitably set up (in matrices). Each sample has a fundamental, transient, x, xx, a weight and two flags. Finally, an event has a dyad, an agent, a client, a behaviour an emotion, potentially a noise term (disrupt flag) and an x observation. The idea is that an agent would actually consist of a full actor that encapsulates all this, and represents the groups it belongs to according to the structure I am showing above. Labeling this structure "groupStruct", then the fullActor would have a groupStruct * array.

gender is handled by simply specifying a dictionary that has the correct sentiments for that gender. so the old interact dictionaries (e.g. `fidentities.dat`) which had male and then female means, the first three are used (male only). To use these for female ratings of identities, they'd have to be split into two dictionaries

4. dyads represent an interaction, and are `dyadIndex` objects which have an index for the dyad, and indices for agent and client in the interaction, an institution list if available, and a random seed.

5. a `samplerState` represents the parameters and samples for each interaction.

   - the parameters are `sentimentParams` which includes all the covariance matrices for the updates and the parameters of impression formation equations (M) and emotion equations (R,E,Q)
   - the samples are an array of sample structures which have 2 double arrays (for f and tau), and then integer indices for agent id, client id, agent behaviour id, client behaviour id and finally a double weight

1.4 Main functions pseudocode

Algorithm 1: FULL STATE UPDATE OVERVIEW.

```
// Main update function (called from bayesactsim)
updateFullStateTwoActorsXX
compute average f,τ
compute optimal interact mean for identities and behaviours (taking partial derivative and setting to zero)
weight→0
for each sample do
    while weight < threshold do
        sampleWeightNextSentimentWithEmotion
          sampleNextState
            if resetting then
                shoehorn in the optimal interact mean for agent or client (depending on reset schedule)
                select schedule by changing a line marked ''RESET_FLAG_MARKER'' in fvarsample.c
                copy out the actual agent behaviour → f_b
            end
            sampleNextSentiment
              setUpSampleNextSentimentRaw
              sampleASentiment // actually does the work of drawing the sample
            updateXSample // update the denotative sample
            if not resetting then
                somaticTransform(WEIGHT,f_a,x_a) // apply Somatic Transforms for identity   - this gives the
                somaticTransform(WEIGHT,f_c,x_c) // for client identity           - sample an initial weight
                somaticTransform(WEIGHT,f_b,x_b) // does not occur if x_b is chosen (reset or getNextAction)
            else
                somaticTransform(DENOTATIVE,f_a,x_a) // extract x_a, x_c from f_a,f_c      - these are the reset
                somaticTransform(DENOTATIVE,f_c,x_c) // for client identity      - (possibly new) identities
            end
    end
    computeSampleWeight // add to sample weights for client emotion and X_x observation if available
end
```

The pseudocode above shows the primary sampling update function and what is called when this update is done. The main sampling update function is

```
updateFullStateTwoActorsXXX
```

which may include $\mathbf{X}$ observations and emotions as well, so there is a series of these functions that can be called with different parameter sets, which would all be one function if C allowed default parameters, which it doesn't.

1.5 Simulation options

The following are the command line options for the *BayesAct* v2 test program `bayesactsim`, which is invoked with:

```
bayesactsim [OPTIONS] simfile
```

where `OPTIONS` are detailed below, and `simfile` is the simulation description file, which contains the description of the interactions you want to simulate. There are a few included in the distribution, to be discussed in Section 1.7 below.

The possible `OPTIONS` are:

- `-h : print usage`
  print the possible options listed here
- `-e : use emotions (default no)`
  To use emotions, specify a -e flag after the function call

  ```
  > ./testbayesact -e simtest-base.txt
  ```

  Note that `-e` flag means that the emotion somatic potentials are not used in the belief updates, but not that emotion values can't be computed based on fundamentals and transients. The `-e` flag is overruled by `-m`, since emotions are not used in INTERACT in an update, although thye can be used to modify identities. Modifying the *BayesAct* code so it can also do identity modification is something to look into.
- `-ep: plan over emotions as well (default no)`
  Emotions will also be used in the plan tree.
- `-x : use additional x state and observations (default no)`
  For applications that have additional state $\mathbf{X_x}$, this flag must be specified.
- `-o <filename >: output a CSV file`
  numerous elements of the state, including possibly the full sample set, are written to a CSV file this can be processed by the script `csvtolatex.py` to get a readable format.

- `-s <num>: num is max number of raw samples to output`
  to the CSV file. If not specified, but -o used, assume none, if zero, assume all)
- `-ga gamma value [0.0025]`
  γ value for MEAN type dictionaries - this overrides `DTYPE_MEAN_VARIANCE`
- `-d : use default action only (default no)`
  Only the default action is used in the plan tree (if used - see `-p` below).
- `-p <num> : use num samples for plan tree generation (default no plan tree)`
  You may want to use a smaller number of samples when planning to get a more well explored, but "rougher" plan tree.
- `-t <num> : timeout (in seconds) for planning (default 0 or 1 rollout)`
  Planning will proceed for this many seconds.

1.6 Python3 wrapper

To use the Python3 wrapper, you have access to only the very basic functions right now, and **these have not been tested recently, so proceed at your own risk.**

- to initialize, you provide the same arguments as to testbayesact in a python3 call: initializeState(numsamples, num_plan_samples, timeout, simfile)
- to get the next action and emotion for the agent for dyad i: (agent_action,agent_emotion) = getNextActionEmotion(i)
- to update the state with the xx observations, the client action and emotion and agent action and emotion as specified (all strings) for dyad i

  ```
  numnodes = updateState(i,xx_obs,client_play,client_emotion,agent_play,agent_emotion)
  ```

  returns the number of nodes in the plan tree - if < 0 this means this failed - something was not in a dictionary (probably an emotion label)

1.7 Usage notes

See detailed explanations in Section 2 for how to specify a set of agents and interactions. A simulation specification has three main sections as follows

- AGENT: specify a set of agents. Each agent is specified with a set of dictionaries (for identities, emotions and behaviours), and two dynamics files (one for impression formation and one for emotions). These are the standard "equations" as used in ACT. Each agent also can have a set of parameters, α_a, α_c, β_a, β_c, δ_a, δ_c and the number of samples used by the agent per interaction. The γ parameters are in the

dictionaries or specified on the command line with `-ga` or in `fvarsample.h` with `DTYPE_MEAN_VARIANCE`

- INTERACTIONS: specify a set of interactions. Each interaction is a dyad (two agents) specified with

  ```
  interaction : A : B
  ```

  this means that this interaction is the specification from agent `A`'s perspective when interacting with agent `B`. To also model agent `B`'s perspective, you need to add a second interaction

  ```
  interaction: B : A
  ```

  Each of the agent's in the dyad is then specified with a probability distribution over identities in the dictionary. Thus, for

  ```
  interaction: A : B
  A : i: p : j : q : k : r
  B : l : s : m : t
  endinteraction
  ```

  means that, from agent `A`'s perspective, agent `A` is a distribution over identities `i,j` and `k` with probabilities `p,q` and `r`, respectively, and agent `B` is a distribution over identities `l,m` with probabilities `s` and `t`, respectively. If no identities and probabilities are specified, then the distribution is even (equal probabilities) over all identities in the corresponding dictionary.

- Institutions are specified per interaction with a single line listing the institutions. These must be in the relevant dictionary.

  ```
  institution: lay : business
  ```

- personas are specified per agent immediately following the agent's name

  ```
  agent Tom : mother : 0.8 : daughter : 0.2
  ```

  means that Tom has a persona that is 80% mother and 20% daughter.

- finally, you can specify which "mode" to run the simulation in either "user" or "events". If "user", then it is interactive and you should make sure you (at least) use the `-v` flag on the command line, and possibly uncomment `VERBOSE` in `fvarsample.h`. If "events", then an events file needs to be specified:

  ```
  events: ../examples/testsim4f.events
  ```

The events file describes a sequence of events across a set of interactions, and is described further in Section 2.

A "user" mode simulation proceeds by

1. asking which dyad you want to simulate - type the index or "enter" to use the same as last time
2. printing out summary statistics for this dyad, including
   - average sentiments (fundamentals and transients)
   - average labels for those sentiments (distribution over denotative states)
   - deflections (expected)
   - deflections (from average state)
   - emotions and labels
3. printing out best next actions for agent
4. asking for the agent's action. "enter" here means that agent does not act.
5. asking for the agent's emotion (if `-e` is specified). Best options are shown.
6. printing out best next actions for client (see (3))
7. asking for the client's action. "enter" here means that client does not act -
8. asking for the client's emotion (if `-e` is specified). A default option is specified.
9. actually doing the update - you have to select the same dyad again to see what happened
10. go to (1)

1.8 Parameters

The parameters that can be changed in the simulation input file are shown in Table 1, and discussed in more detail in Section 2. Here I remind users of the meanings of these parameters, followed by the meanings of a few other settable constants (mostly in `fvarsample.h`).

- α_a, α_b and α_c: the strength of the affect control principle. If higher the ACP is weaker.
- β_a and β_c: the identity sentiment inertia - how much do we expect sentiment to stay the same over time.

- δ_a and δ_c: the denotative identity inertia - how much do we expect denotative labels of identities to stay the same over time. If working a simulation close to equilibrium (everything stays low deflection), then this can be set small (and possibly the `RESET_THRESHOLD` large. If working in off-equilibrium situations (e.g. something surprising happens), then this must be larger (and possibly the `RESET_THRESHOLD` correspondingly smaller).

- n is the number of samples to be used by the agent for each of its interactions.

- γ is specified by default in `DTYPE_MEAN_VARIANCE` in `fvarsample.h`. You can also specify it command line using `-ga`. However, these only work for dictionaries of type `MEAN`. Dictionaries of type `SD` or `COV` will have the variances in the dictionary, and these variances may be different for each dictionary item. Further, as the dictionaries are specified for self and other, each agent can use two sets of variances: for identities related to the self, and for identities related to the other.

The items settable in `fvarsample.h` are:

- `RESET_THRESHOLD`: the threshold at which a reset is triggered. This threshold is on the total weight of all samples. It can be as small as 10^{-307}. Setting this number, especially in relation to the setting of δ above, is important to how the denotative identity distributions unfold over time.

- `DO_INTERACT`: Whether to compute interact updates as you go - this is used for resetting, so is normally on

- `PERSONA_FRACTION`: how many samples are added from the persona at each step

- `DO_DENOTATIVE_IDENTITY_UPDATES`: if 0, will keep denotative identities constant (except for resets)

- `VERBOSE` and `VERYVERBOSE`: different settings for verbosity

Finally, two items settable in `bayesactsim.h` are:

- `DISRUPT_NOISE`: the amount of noise added if a ``+'' is specified in the events file

- `WRITE_IMAGES`: whether to write out small images showing the samples at each step. Set `image_type` according to which two dimensions of sentiment you want to see if the images (default E and P).

- `SPECIFY_ALL_FOUR`: if 1 then when querying in "user" mode, all four actions are queried for.Usually only one action is required, the others being none,none, and the same action as taken, but noise can change this, so this option may be needed.

2 Example Simulations

In this section I run through a few simple simulations. First, in Section 2.1, one with a single dyad with unimodal distributions that follow an equilibrium situation, as in interact. Second, Section 2.2 with a single dyad that does some re-identification. Section 2.3 then shows how we can closely imitate Interact, and Section 2.4 shows a more involved situation involved three people. Finally, Section 2.5 shows the example that is covered in more detail in (Hoey and Schöder, 2021).[1]

Throughout these simulations, we remind the reader that there are two methods for computing somatic transforms. The *averaged* method is more suited to computations that try to mimic INTERACT, while the *expectation* method is a more comprehensive and "correct" method. We use the *expectation* method in the following.

Note that we will be simulating *BayesAct without* the addition of the additional denotative state, $\mathbf{X_x}$. Thus, there is no reward, and no point to using the planning capabilities of *BayesAct*.

You can run a simulation in your browser now by heading to `cs.uwaterloo.ca/~jhoey/research/bayesact/code/index.html`. The simulation corresponds to the one discussed in Section 2.5. It also allows you to download the code and the simulation file you used. We plan in future to open up the *BayesAct* server so simulation files and settings can be submitted online.

2.1 Single Dyad, close to Interact

Simulation specification: `simtest-base.txt`

In this simulation, we use the Indiana 2002-2004 databases and dynamics equations. These can be obtained from INTERACT (see below). We will be simulating two males, "Hank" and "Tom". Hank is Tom's *secretary*, and Tom is Hank's *boss*, and both share the exact same representation of the situation.[2] The simulation is done using the default α, β parameters (specified in `fvarsample.h`) and the γ parameter is in the input identity file or taken as `DTYPE_MEAN_VARIANCE`, also in `fvarsample.h`. The default values are as follows In this first simulation, we set `DTYPE_MEAN_VARIANCE` to 0.01 in order to ensure the sentiments are interpreted as they would be in Interact. However, as we will see this causes problems for the sampler if the interaction leaves the equilibrium it is on (if deflection goes too high).

We first specify our two agents in terms of their somatic potentials they will use (their sentiment dictionaries). We use dictionaries of type `MEAN` as the Indiana 2002-2004 database does not have variance information.

[1] available from the author on request.

[2] We use the term "secretary" here in full recognition that it is outdated and no longer in common use. We use the term here, and any genderedness of the proper names we are using (Hank and Tom), without prejudice and these could be replaced by any names.

param.	code name	hard code	com line	sim file	default value
α	DEFAULT_ALPHA	fvarsample.h	-ba,-bc	alphas: [a or aa : ab : cc]	0.1
β	DEFAULT_BETA	fvarsample.h	-ba,-bc	betas: [b or ba : bc]	0.01
δ	DEFAULT_AGENT_DELTA	fvarsample.h		deltas: [d or da : dc] ...	0.1
γ	DTYPE_MEAN_VARIANCE	fvarsample.h	-ga	as SD/VAR in data files	0.1
n	num_samples	1000			
r_t	RESET_THRESHOLD	10^{-32}			

Table 1: Modifiable parameters in *BayesAct*.

Our first agent is Hank. His specification starts with the keyword `agent:` and his name on a line, followed by four dictionaries and two equations files, followed by the keyword `endagent`. All fields are separated by colons (`:`).

```
agent: hank
dictionary: AGENT : fidentities.dat : MEAN
dictionary: BEHAVIOUR : fbehaviours.dat : MEAN
dictionary: CLIENT : fidentities.dat : MEAN
dictionary: EMOTION : fmodifiers.dat : MEAN
dynamics: IMPRESSION : tdynamics-male.dat
dynamics: EMOTION : temotions-male.dat
endagent
```

The files `fidentites.dat`, `fbehaviours.dat` and `fmodifiers.dat` come with the *BayesAct* package, and are the Indiana 2002-2004 ACT lexicons. They can be obtained from INTERACT by going to the `Import/Export` tab, selecting `Show current entries` and then cutting-and-pasting into the corresponding text file. Other dictionaries could be used in the same fashion. The files `tdynamics-male.dat` and `temotions-male.dat` are the impression formation and emotion formation equation parameters, obtained from INTERACT by doing to the `View equations` tab, selecting `USA 1978` and `Male Actor-Behaviour-Object`.

Our second agent is Tom, and he is using the same somatic potentials as Hank.

```
agent: tom
dictionary: AGENT : fidentities.dat : MEAN
dictionary: BEHAVIOUR : fbehaviours.dat : MEAN
dictionary: CLIENT : fidentities.dat : MEAN
dictionary: EMOTION : fmodifiers.dat : MEAN
dynamics: IMPRESSION : tdynamics-male.dat
dynamics: EMOTION : temotions-male.dat
endagent
```

Now, we specify the interactions between these two. Each starts with the keyword `interaction`, followed by the names of the two agents in the dyad. The first agent named is the one from whom's perspective this interaction is modeled from. Thus,

```
interaction: hank: tom
```

means we are specifying how *hank* is viewing his interaction with *tom*.

```
rseed : 1621517535849
```

I am using specific random number seeds to the random number generators give the exact output you see below. If you comment out these lines, you will get a slightly different simulation. The more samples you add, the smaller the variation will be.

```
interaction: hank : tom
hank : secretary : 1.0
tom : boss : 1.0
endinteraction
```

In this case, *hank* sees himself as a *secretary* with probability 1.0, and he sees *tom* as his *boss* with probability 1.0. Tom's view of the situation is identical:

```
interaction:  tom: hank
tom : boss: 1.0
hank : secretary : 1.0
endinteraction
```

The second interaction can be left out of the file, in which case only the first interaction will be run. However, Interact cannot be used in this case, so `DO_INTERACT` must be false.

Finally, the file shows two more things that can be set: the number of interations to simulate for, and whether to query the user for the simulation `simtype: query` or to run automatically `simtype: bayesact`.

To set this up in INTERACT, define two interactants (*hank* and *tom*), then define the situation as a secretary and boss, and then define single event `hank[_,secretary],dress,tom[_,boss]`, as `dress` is the most aligned behaviour for *hank* to take first. Go to `select options` and get everything written out to the Java Console.

A *BayesAct* simulation can be run from the `source/` directory as follows:

```
./bayesactsim testsim-interact.txt
```

You can edit `fvarsample.h` and modify some settings, such as the default number of samples, etc. These can also be modified in the simulation file (e.g. `simtest-base.txt`). You can also switch on VERBOSE to see even more output to the screen, or you can go to `bayesactsim.c` and uncomment WRITE_IMAGES to see a simple visualization of the emotional state for each frame dumped to `../output/`.

I will walk through this in USER mode. If you specify an "events" file, you can run this in batch mode, see `testsim.events`, explained in the next simulation. First, the program

spits out a bunch of information about the different dictionaries (if `VERBOSE` is `define`-d in `fvarsample.h` , then displays:

```
available dyads:
0: hank - tom
1: tom - hank
select dyad to simulate (enter for same as last time 0):
```

Press enter to select dyad 0, which means we'll be doing a simulated step from Hank's perspective. This is like selecting `Experiences of hank` in the `define events` tab in Interact. Now, the program outputs the current situation, including the fundamentals and transients (which are the same as fundamentals since its the first step):

```
fundamental output average
ae      ap      aa      be      bp      ba      ce      cp      ca:
1.03    0.01    -0.18   0       0       0       0.48    2.16    0.94


transient output average
ae      ap      aa      be      bp      ba      ce      cp      ca:
1.03    0.01    -0.18   0       0       0       0.48    2.16    0.94
```

here each row shows agent, behaviour and client (actor, behavior and object in ACT) values for E,P, and A (e.g. the "ce" term is the client's "E" value). These are exactly those from the dictionary (e.g. *boss* is 0.48, 2.16, 0.94) as it is the first step. Next, the distribution over denotative identities corresponding to those are shown. These are computed with the *expectation method* only, and in this case each agent has a single identity with probability 1.0:

```
client identity average labels (probabilities):
boss (   1)

agent identity average labels (probabilities):
secretary (   1)
```

The deflections are all zero since transients are the same as fundamentals, and the emotions are characteristic since transients and fundamentals are the same. The emotions are shown using one of two methods, described as follows, and specified as `ae_method` in `bayesactsim.c`: either `AVERAGED` or `EXPECTATION`.

The (*averaged method*) works by computing the expected fundamental and transient sentiments (these are the same as those shown above), and then applying the emotion formation

equations to get the EPA for the emotion resulting from the averaged sentiments, and then applying a somatic transform to find the closest label (this method is the one that is in correspondence to INTERACT). In the second (*expectation method*) the expected emotions are computed directly (applying the emotion formation equations and somatic transform to each sample and gathering statistics on the labels that are found). I am using the *expectation* method in the following simulation. Using this method, we find *warm* taking all the probability. In this case, the sentiments for *warm* are (EPA:$\{1.72, 0.57, 0.06\}$). Compare to INTERACT's computation of the *characteristic emotion* `feeling effects` tab, which is (EPA:$\{1.61, 0.48, 0.38\}$).

The averaged emotion for *tom* is (EPA:$\{1.6, 2.7, 1.18\}$) is close the the *characteristic emotion* (use the `feeling effects` tab in INTERACT) of (EPA:$\{1.61, 3.01, 1.21\}$), and the expected emotion is a single one *contented* (with probability 1.0).

Finally, the best action to take for *hank* is shown, again shown in the way as specified by `ae_method`. The (*expectation method*) computes the distribution over denotative labels (the expectation of the denotative state), second (*averaged method*) computes the best action based on the expected fundamentals and transients (the denotative state corresponding to the expected sentiments, again this is the one that should correspond to INTERACT):

```
got 404 potential actions for hank (from perspective of hank), top 5 are
rank *  cum. prob  label (probability)
-----------------------------------------------------------------------
1 * 0.931 dress (0.931)
2 * 0.997 turn to (0.0658)
3 * 0.999 wait on (0.00209)
4 *    1 pet (0.000483)
5 *    1 watch (0.00032)
=======================================================================
```

And we see that Hank would likely *dress* his boss, Tom. The program also shows that Hank expects Tom to *glorify* 98%, or *discipline* (2%).

To get the same numbers from INTERACT, head to the `feeling effects` screen, select `secretary` on the left and `boss` on the right, click `characteristic emotion` for both, then click `compute solution` making sure that `behaviour` is selected. You'll get an expected value of `1.1 -0.35 -0.40`, which is very close the value above, with the same denotative label *dress*.

We enter `dress` for Hank's action (to correspond to Interact), and then we see the best actions to take for *tom*.

Now select dyad 1 and we can go and see what effect this dressing had on Hank's boss, Tom:

```
fundamental output average
ae ap aa be bp ba ce cp ca:
0.476 2.15 0.936 1.02 -0.0331 -0.341 1.03 0.0129 -0.18

transient output average
ae ap aa be bp ba ce cp ca:
0.375 1.35 0.567 0.568 -0.0327 -0.0962 0.809 -0.165 -0.0877

client identity average labels (probabilities):
secretary (0.998) homemaker (0.002)

agent identity average labels (probabilities):
boss (   1)
```

Indeed, he certainly still feels like a *boss* and thinks Hank is his *secretary*. In Interact, we run the interaction and get the following output for fundamentals (in the same order as shown for Tom above:

```
0.48   2.16   0.94    0.95   0.07  -0.33   1.03   0.01  -0.18
```

And for transients:

```
0.36   1.31   0.57  0.53   0.05  -0.09  0.77  -0.12  -0.08
```

The correspondence is close. *BayesAct* then outputs the deflections, again computed in two ways as the expected deflection (average deflection computed per sample - *expectation method*):

```
deflection AGENT: 0.13
deflection BEHAVIOUR: 0.314
deflection CLIENT: 0.87
total deflection:  1.14
```

showing numbers that are very similar to INTERACT:

```
Deflection:   1.19,  Actor:   0.09,  Object:   0.87.
```

Emotion and action information is shown next. I will focus here only on action distributions as the emotion model is still a work in progress. The emotions are printed to the screen for your viewing, but do not play a role in the simulation unless you add the `-e` flag when running `bayesactsim`

The best actions to take for Tom are

```
======================================================================
got 397 potential actions for tom (from perspective of tom), top 5 are
rank *  cum. prob  label (probability)
----------------------------------------------------------------------
1 * 0.996 discipline (0.996)
2 *    1 examine (0.00362)
3 *    1 face (3.25e-09)
4 *    1 employ (1.59e-09)
5 *    1 march with (1.58e-09)
======================================================================
```

which is again in close correspondence to INTERACT (discipline): 0.93, 2.3, 0.54 Now we can select a behaviour for Tom and then continue from here. Let us see, however, what happens if things go off the rails and Tom does something unusual and unexepected, like "punch" Hank. The following tragic information shows up:

```
number of reset attempts: 1000 successes: 0
number of resets for agent: 0 client: 0 both: 0
total weight of all samples in update 0
total weight of all samples for agent in update 0
total weight of all samples for client in update 0
```

followed by this very subtle and self-effacing error-like message:

```
----reset failed: weight 0 threshold 1e-32.
```

This message is devastating in its simplicity, and may lead to dispair and discouragment on your part, but I urge you to take heart, for it only means that the sampler broke. What should you do? First, you should probably not trust your results as the distribution is essentially just copied over from the previous time step, such that all intervening evidence (suck as the "punch" above) is ignored. Second, you should read the next section.

2.2 Why did this happen and how can we fix it?

sim filename: `simtest-flex.txt`

The default parameters expect things to be *roughly* close to equilibrium. Thus, "punching" is just too far away from any reasonable definition of a boss-secretary relationship, that the sampler just can't handle it. You have two options here. The first is to increase the number of samples by adding this to each agent definition in the simulation file:

```
numsamples: 10000000000 // or whatever you want
```

However, I cannot make any guarantees on how big this number will need to be for your specific situation, and therefore how long a simulation will take for your case. I have tried the above with a million samples (instead of 1000) and not done any better. Luckily, you have a second option, which is to make the agent more flexible in what it defines as its possible identities. This makes use of the "RESET" mechanism in *BayesAct*, which works as follows.

When the sampler breaks (as above), *BayesAct* will attempt to re-identify the client, followed by the both agent and client. It does this by first finding the zero-point of the partial derivative of the deflection, as Interact does, and then setting the sentiments for the *BayesAct* agent to a distribution centered on that point for the identity in question. It is only when these two attempts fail that *BayesAct* complains with the `----reset failed` message.

The simulation we did above did not "make use of" the reset capabilities of *BayesAct* because α and δ are too small. These control the spread of the agent's distribution in both connotative (α) and denotative (δ) spaces. We will also increase β (which is normally 1/10 of α).

We therefore write:

```
agent: hank
alphas: 1.0 : 1.0 : 1.0
betas: 0.1 : 0.1
deltas: 0.5
...
```

and similary for Tom. Note the `alphas` and `betas` are specified for agent, behaviour (for `alphas`) and client, but if they are all the same, they can be specified as just as single number as `deltas` are.

Running the same simulation as above (Hank `dresses` Tom, and then Tom `punches` Hank, we no longer run into the terrible `-----reset failed` message, and we get the following

```
tom is updating and perceives him/herself doing [punch] to hank,
                            who is doing [[doesn't act]] back
number of reset attempts: 751 successes: 291
number of resets for agent: 0 client: 20 both: 271
total weight of all samples in update 0.00877612
hank is updating and perceives him/herself doing [[doesn't act]] to tom,
                            who is doing [punch] back
number of reset attempts: 740 successes: 622
number of resets for agent: 0 client: 596 both: 26
total weight of all samples in update 0.000294339
total weight of all samples for agent in update 0
total weight of all samples for client in update 0.000294339
```

Now we see that of 1000 samples for Tom, 751 of them attempted a reset, of which 291 succeeded. Of these 291, 20 succeeded after only re-identifying the client, while 271 of them required a reset of both agent and client. For Hank, the split was 596/620 successful resets for client only, meaning Hank had an easier time reconciling the punch with his own identity of self, `secretary`. Thus, we expect Hank remains `secretary`, but re-identifies Tom, while Tom is going to re-identify both.

Selecting dyad 0 this time, we see how Hank is handling all this

```
ae ap aa be bp ba ce cp ca:
1.05 0.166 -0.253 -1.45 0.192 1.44 0.307 2.31 1.03

transient output average
ae ap aa be bp ba ce cp ca:
0.113 -0.877 -0.048 -1.25 0.595 1.04 -1.08 0.919 0.952

client identity average labels (probabilities):
VIP (0.992) boss (0.006) hunk (0.001) celebrity (0.001)

agent identity average labels (probabilities):
homemaker (0.992) woman (0.005) egghead (0.001)
          stuffed shirt (0.001) questioner (0.001)
```

Now we see Hank has re-identified Tom as a `VIP`, and himself as a `homemaker` (which is close to `secretary` in this dataset. Tom on the other hand has re-identified himself as perhaps a `bouncer` and Hank as a `foster child` or a `tenant`. Hank's optimal action would now be to *reproach* Tom, and the simulation can continue from here.

2.3 Interact: closing in

simulation file: `simtest-interact.txt`

In this simulation we attempt to get *BayesAct* to reproduce what Interact produces. We will see how this can also break the simulation and how to resolve it.

There are a few things to note about this exercise. First, *BayesAct* is fundamentally computing a different posterior update based on the total derivative of the deflection, while Interact is computing partial derivatives. Therefore, *BayesAct* really is computing a different solution than Interact, and we should not expect the same results. Nevertheless, we can produce a version of *BayesAct* that is as deterministic as possible by using the following parameter settings:

- $\alpha = 0.01$ set in `simtest-interact.txt`
- $\beta = 0.001$ set in `simtest-interact.txt`
- $\delta = 0.01$ set in `simtest-interact.txt`
- $\gamma = 0.01$ set using `DTYPE_MEAN_VARIANCE` in `fvarsample.h`
- `RESET_THRESHOLD`: 10^{-307} set in `fvarsample.h`
- `num_samples`: 10000 set in `simtest-interact.txt`

re-running the same simulation as above again, we see largely the same results as with the default parameters. When the deflecting action of *punch* happens, however, the samplers again complain that `---reset failed`.

Once again, the ability to handle the deflection action requires the agent to be somewhat more flexible in its choices of identities at the denotative level. This can be fixed by setting `DO_DENOTATIVE_IDENTITY_UPDATES` to 0 in `fvarsample.h`, or by increasing δ in the simulation file. if we set δ to be 1.0 in `simteset-interact.txt`, the simulation can once again proceed with some degree of accuracy. However, note that *BayesAct* would be better off running this simulation with the default parameters shown at the start of this section, as these constricted versions we are using here pose too many constraints for the *BayesAct* model to handle properly.

2.4 Threesome

Simulation specification: `testsim-threesome.txt`

In this simulation, we see a variety of other possibilities. First, look at the bottom of the file, you see

```
events : ../examples/testsim-threesome.events
simtype : events
```

This indicates the simulation will proceed according to the script in the file shown. This file looks like this

```
tom : * : : sally : : :
sally : * : : tom : : :
sally : * : : dave : : :
dave : * : : sally : : :
```

It indicates a series of events with the actor in the event first, following by the actor's action ("*"=optimal, "!"=Interact," "=none,"[label]"=dictionary word). Thus this events file says that tom and sally each exchange optimal actions, followed by sally and dave.

Returning to the start of the file, we now see that each agent can specify its values fof α, β and δ (γ is in the dictionaries or is specified as `DTYPE_MEAN_VARIANCE`).

There are a number of new elements in the interaction specification. First, should institutional labels be in the dictionaries, you can write a set here. Within this interaction, the acting agent will only maintain indentites/behaviours that agree with these instiutional labels. Further, each agent can have a distribution over identities for self and other, as shown. Thus, in this example, dave is sally's *husband*, and part-time *father* when interacting with his children, but he also feels she is a *lady* and that he is sometimes a *loser*.

```
interaction: dave: sally
institution : family  :lay
dave : husband : 0.8 : father : 0.1 : loser : 0.1
sally : wife : 0.5 : lady : 0.5
endinteraction
```

You can track and see what each dyad is doing in the simulation. I will not go through this in detail here, but will note that you can change this to "user" mode and do it interactively.

2.5 Multiple Identities

Simulation specification: `testsim4f.txt`

This is the simulation from the paper (Hoey and Schöder, 2021) describing an environmentalist who thinks a coal miner may be somewhat to blame for a climate crisis, and the coal miner can feel the environmentalist is prosecuting them. However, they can settle on a joint, shared, identity of *citizen*, should they be able to align properly. The simulation file uses a set of dictionaries of type `SD`, meaning the ***variances*** of the denotative elements in the dictionary are specified therein. The events file also contains a new symbol "+" which means to disrupt the incoming observation to the action in question by an amount `DISRUPT_NOISE` (in `bayesactsim.c`).

```
hank : *+ : * : tom : + : *:
tom : *+ :* : hank :+ :* :
```

In this case, both agents have their observations of the other's action disrupted by some noise. The disruptive noise is implemented at this point as follows. Clearly, it could be implemented in many different ways depending on the noise level and domain (e.g. in a real robot, in might be real nenvironmental noise). First, we translate the behaviour that was taken by the other agent into EPA space, then we add noise as a draw from a random Gaussian with standard deviation `DISRUPT_NOISE`, and then translate back to a denotative label using a somatic transform. The result may be the same as without the noise.

In this case, we will simulate the complete set of interactions in `testsim4fne.events` which are 9 mutual exchanges bewteen the two actors, each acting optimally, each with `DISRUPT_NOISE=0.1`. We output the result to a csv file using

```
./bayesactsim -o ../output/testout4f.csv ../examples/testim4f.txt
```

which we can convert to a readable latex table using the script `output/csvtolatex.py` in from the `output/` directory, we can then do:

```
python3 csvtolatex.py ./testout4f.csv ./colstolatex ./colstolatexi
                >& coalminer2.tex
```

Which gives the data shown in Table 2. We see in this case that the two agents have difficulty aligning. The parameters can be set in such a way as to make them more likely to align. See (Hoey and Schöder, 2021) for more details.

References

Hoey, J., MacKinnon, N., and Schröder, T. (2021). Denotative and connotative control of uncertainty: A computational dual-process model. *Judgment and Decision Making*, 16(2).

Hoey, J. and Schöder, T. (2021). Disruption of status orders in societal transitions as affective control of uncertainty. *under review*.

MacKinnon, N. J. and Hoey, J. (2021). On the inextricability and complementarity of cognition and affect: A review and model. To appear, *Emotion Review*.

Hank				Tom			
d	hank id p	tom id p	behav. ↔	tom id p	hank id p	d	Δf
0.0	0.5 citizen 0.5 environmentalist	0.5 citizen 0.5 culprit	inspect →	0.5 citizen 0.5 coal miner	0.5 citizen 0.5 prosecutor	0.0	3.9
3.0	0.7 environmentalist 0.1 peer	0.8 culprit	confer with ←	0.3 miner 0.2 coal miner 0.1 laborer 0.1 citizen 0.1 colleague	0.6 prosecutor 0.1 citizen 0.1 colleague	1.1	3.7
5.7	1.0 environmentalist	0.9 malcontent	test →	0.3 citizen 0.3 colleague 0.1 peer 0.1 wage earner	0.3 citizen 0.2 colleague 0.2 wage earner 0.1 peer	1.5	3.7
7.4	0.7 environmentalist 0.1 peer	0.6 malcontent 0.3 culprit	confer with ←	0.3 citizen 0.3 colleague 0.1 peer 0.1 wage earner	0.3 citizen 0.3 colleague 0.1 wage earner 0.1 peer	1.9	3.7
8.1	0.6 worker 0.2 peer	0.7 sinner 0.2 malcontent	pacify →	0.3 citizen 0.3 colleague 0.2 wage earner 0.1 peer	0.3 colleague 0.3 citizen 0.1 wage earner 0.1 peer	2.0	3.6
8.3	0.4 environmentalist 0.2 worker 0.2 peer	0.5 malcontent 0.3 sinner 0.2 culprit	confer with ←	0.3 colleague 0.3 citizen 0.1 wage earner 0.1 peer	0.4 colleague 0.3 citizen 0.1 wage earner	2.0	3.6
			...				
7.3	0.9 environmentalist	0.9 malcontent	pacify →	0.3 colleague 0.3 citizen 0.1 wage earner 0.1 peer	0.4 citizen 0.3 colleague 0.1 wage earner 0.1 peer	1.9	3.3
7.0	0.6 environmentalist 0.2 peer 0.2 worker	0.7 malcontent 0.2 sinner	confer with ←	0.4 citizen 0.3 colleague 0.1 wage earner 0.1 peer	0.4 citizen 0.3 colleague 0.1 peer 0.1 wage earner	1.9	3.3

Table 2: The distributions over denotative labels are cut off when the cumulative probability exceeds 0.6. d= deflection, Δf fundamental difference (smaller is better). There are 10 iterations in the middle left out. Here the agents have default parameters and to not align well. See the ABS paper for details.