Documentation: DASH socog belief module

Nathaniel Rodriguez

1 Model

The original opinion model in [1] defined two networks: (1) a social network, and (2) a belief network. The social network is defined by agents who are nodes and undirected, unweighted relationships between nodes, which could denote exposure or friendship ties. In the belief network, nodes are concepts and edges are undirected, weighted relationships between concepts. These relationships represent a valence or association between concepts that can be positive or negative. A pair of concepts and the corresponding link between them constitute a *belief*. A triple of concepts and their corresponding links constitute a *triad*. In our model, beliefs are transmitted between agents along social network ties. Beliefs are accepted or rejected based upon a set of rules that factor in the *coherence* of the whole belief network and the social influences of neighbors. There are two energy terms, one for internal beliefs and one for social pressures. The sum of these two become the total energy. An agent accepts a belief if their total energy decreases upon its acceptance, else the belief is accepted with a probability inversely proportional to the energy difference between the agents current state and the candidate state.

The internal energy is derived from the total stability of *triads* in the belief network in a fashion akin to social balance theory:

$$E_n^{(i)} = -\frac{1}{\binom{M}{3}} \sum_{j,k,l} a_{jk} a_{kl} a_{jl}, \tag{1}$$

where M is the number of nodes in the belief network and a_{jk} is the association connecting nodes j and k, which can be positive or negative. The sum is taken over all triads in the belief network and normalized by the total number of triads.

We represent the society as a social network, $\mathcal{N}$, where $N = |\mathcal{N}|$, and whose nodes are individuals and edges represent social relationships through which ideas are communicated. We define the social energy term to be the degree of alignment between connected individuals. The local social energy that an individual $n \in \mathcal{N}$ feels can be defined by:

$$E_n^{(s)} = -\frac{1}{k_{\max}\binom{M}{2}} \sum_{q \in \Gamma(n)} \vec{S}_n \cdot \vec{S}_q, \tag{2}$$

where the sum is taken over the set of n's neighbors in $\mathcal{N}$, denoted by $\Gamma(n)$. $\vec{S}$ is a belief state vector where each element corresponds to an edge in the belief network, so $|\vec{S}| = \binom{M}{2}$. $k_{\max}$ is a normalization constant that bounds the strength of peer-influence and is equal to the maximum degree of $\mathcal{N}$.

The original formulation of the model in [1] was made for a synchronous environment with perfect information. An agent has complete knowledge of its own belief system as well as its neighbors. The network was also explicitly defined so each agent also knew who its neighbors were. However, in the DASH framework, the simulation is asynchronous, information can be imperfect, an agent may not have an explicitly formed social network, the social network maybe a multigraph with many different types of edges, the neighbors and their beliefs maybe expensive

to access, or the neighbors and their beliefs maybe unrealistic to access from a behavioral standpoint. Additionally, the agent may not get all its information from its neighbors, but may observe features in its environment that contribute to its beliefs.

In order to take these issues into account I have created an adaptation of the original cognitive-social belief model. In equation 2, a dot product is taken over the agent's belief vector and its neighbors. A sum of these products was taken to get a value for how far the agent's beliefs were from its neighbors. We can move the dot product out of the sum, and let the sum represent a new vector that I will call the *perceived belief vector*:

$$E_n^{(s)} = -\frac{1}{k_{\max}\binom{M}{2}} \vec{S}_n \cdot \vec{S}^{(p)} \tag{3}$$

The perceived belief vector is an agglomeration of all the incoming beliefs that the agent receives. While we explicitly abandon the network structure from the equation, in practice, the perceived belief vector can represent an approximation to the agent's neighborhood. In order for this to be the case, we need an update equation for the perceived belief vector so that new information about beliefs can be incorporated into the agent's perception:

$$S_i^{(p)} \leftarrow S_i^{(p)} + \frac{1}{\tau}\left(S_i - S_i^{(p)}\right) \tag{4}$$

At each update step, when the agent receives a new belief S_i, it will be added to the current perceived belief vector. Perceptions are only updated when new belief information is acquired. Older information about beliefs will eventually get crowded out as new belief information overwrites it. The time-constant τ controls how fast this happens. Time is omitted because the update only occurs when a new belief is received. An agent that is isolated from the environment will retain a memory of what people's beliefs were before. Only active agents will be able to keep up-to-date with other's beliefs through interaction with the environment. The belief update does not depend upon the source. This equation drives the agent's perceived beliefs toward the incoming belief.[1]

In order to address variable size belief systems and unknown neighbor information, the normalization constants are removed from the energy equations so that:

$$E_n^{(i)} = -\sum_{j,k,l} a_{jk} a_{kl} a_{jl}, \tag{6}$$

and

$$E_n^{(s)} = -\vec{S}_n \cdot \vec{S}^{(p)} \tag{7}$$

The total energy is still a weighted sum of the individual energies:

$$H = \left[JE_n^{(i)} + IE_n^{(s)}\right] \tag{8}$$

[1] Alternative update equation. Is a shunting equation, drives beliefs to the extreme of whatever direction they point. Would result in perceived beliefs representing a more extreme perception of actual neighbor beliefs.

$$S_i^{(p)} \leftarrow S_i^{(p)} + \frac{1}{\tau}\left[\frac{\max\left(S_i, 0\right)}{V}\left(V - \max\left(S_i^{(p)}, 0\right)\right) + \frac{\min\left(S_i, 0\right)}{V}\left(V + \min\left(S_i^{(p)}, 0\right)\right)\right] \tag{5}$$

The valence is the bound between $[-V, V]$ and τ is a time-constant that adjusts the strength of the contribution of a new belief. This is a double bounded shunting equation which keeps the valences of $S_i^{(p)}$ within $[-V, V]$.

And the belief is still accepted if the change in energy passes a check determined by:

$$P(\text{accept}) = e^{\frac{-\Delta H}{T}} \tag{9}$$

where T acts like a temperature for beliefs.

2 Socog Module

Documentation for the socog module and tutorials (taken from doc strings, can use `help(ClassName)` for more information:

2.1 Concept

An immutable object representing a concept with `name` and `id`. name and id are not actually used here in the module. Any object that is hashable, immutable, and has `__eq__`, can be used as a concept.

2.2 ConceptPair

An immutable object representing a pair of concepts. Concepts can be any hashable or immutable type with `__eq__` defined. Elements can be independently accessed but is still hashable and order doesn't matter:

E.g. `ConceptPair(concept1,concept2) == ConceptPair(concept2,concept1)`

ConceptPairs are used as keys for Beliefs.

2.3 ConceptTriad

An immutable object that represents a triangle, of three concepts. Concepts can be any hashable or immutable type with `__eq__` defined. Internally it is represented as 3 ConceptPairs. Elements can be independently accessed but are still hashable and order doesn't matter:

E.g. `ConceptTriad(concept1,concept2,concept3)`
`== ConceptTriad(concept2,concept3,concept1)`

ConceptTriads are a convenient way to store concept-pair and concept information for calculating the internal energy, checking for triangles, or looking-up specific Beliefs.

2.4 Beliefs

An object that represents a set of beliefs. It also doubles as a vector. Currently implements dot product. If needed, add support for other math operations.

There are no 'scalar' beliefs, so a single belief is represented as a single element vector.

`Beliefs` subclasses `dict`, so it has access to all the functions `dict` provides and can be instantiated in the same fashion.

2.5 BeliefNetwork

Nodes represent concepts in the belief network and edges represent a valenced relationship between those concepts. The valence can be positive or negative. No connection means no relationship. Triads of beliefs are checked for stability: `+++` is stable, `-++` is unstable, `--+` is stable, `---` is unstable. Edges can be weighted (e.g. valence). The product of the weights specifies triad stability. Given a set triad with valences a_i, a_j, and a_k, the energy is given by: triad energy $= -a_i a_j a_k$

Lower energy corresponds to higher coherence of beliefs.

A BeliefNetwork can be modified in-place via addition with another belief networks or Beliefs through the `+=` or `-=` operators. Scalar values can also be add/subtracted from the valences of the entire vector. Dot product is also supported with `*`. However, `*` used with a scalar carries out normal multiplication. Add additional operators if needed.

The energy of the belief network is the internal energy, not to be confused with the social energy.

2.6 BeliefModule

Represents a cognitive module for assessing the coherence of an agents belief system. It handles receiving (`process_belief`) and sending (`emit_belief`) beliefs, updating the belief network of the agent, and determining whether to accept or reject new beliefs.

Shallow copy access to the underlying beliefs of both the beliefs and perceived beliefs is given via the `beliefs` and `perceived_beliefs` properties. Alternatively, direct access to those items can be achieved via `obj.belief_network.beliefs` or `obj.perceived_belief_network.beliefs`

A BeliefModule can be instantiated with the following parameters:

- `belief_net`: a BeliefNetwork type object. $default$ =`BeliefNetwork()`
- `perceived_net`: a BeliefNetwork type object representing the agent's perception of other's beliefs. $default$ =`BeliefNetwork()`
- `seed`: seed number for internal rng (uses Random). $default = 1$
- `max_memory`: how many steps back the agent remembers incoming beliefs. This can be used in methods for outputing recent beliefs. $default = 1$
- `T`: Higher T will increase the likelihood of accepting beliefs that would increase the agents energy. $default = 1.0$
- `J`: coherentism, controls contribution of internal belief system energy to total energy. $default = 1.0$
- `I`: peer-influence, controls contribution of social energy to the total energy. $default = 1.0$
- `tau`: float between $[1, inf]$. Higher values prefer older beliefs and reduce the contribution of newer beliefs to perceived belief network. $default = 1.0$
- `recent_belief_chance`: $[0, 1]$, the probability of choosing a belief to emit from short-term memory. If a recent belief isn't chosen, then a belief is chosen uniformly at random from the belief network. $default = 0.0$
- `verbose`: prints additional information

2.7 SocogSystem1Agent

A system1-like class that uses a belief module and system rules to fill a queue of actions that the agent will attempt to take. It can be instantiated with the following parameters:

- `belief_module`: A `BeliefModule` object

For the modeller, the most important method is the `read_system1_rules` method which allows the user to define the rules for system1. This method should be called when subclassing the `SocogDASHAgent`. A rule string is given as input, similar to how system2 receives goals. However, system1 uses a list of conditions that are evaluated. If a condition is true, then a corresponding sequence of actions will be added to a queue of actions in the same order as the

conditions. If an action fails, the sequence is reset[2]. Conditions are re-evaluated once the queue has completed or if `process_belief` is called. To prevent recursion, if `process_belief` is called within the conditions it will not prompt a reset.

Whenever actions are taken, variables are stored internally. Whenever the sequences completes or fails the variable bindings are reset. Because it is impossible for the class to know when a primitive is supposed to set or use variables, it is up to the user to write primitives that check `isVar`.

The parameter `rule_string` is a string containing lines for each system rule. A system rule is defined by a condition and sequence of actions. If the condition is fulfilled, the sequence is carried out. The condition is a logical statement that can use and, or, not, and parentheses. The action statement is a sequence of primitive actions separated by spaces. The conditions can specify beliefs, which require three values to be specified: two concepts and a valence. Or the conditions can specify primitive actions. For example:

```
if [A,B,0.5] and [C,B,-0.2] then run(speed) jump(height)
```

Actions can also be added to conditions like so:

```
if foo(Belief) and fee(Belief,height) then jump(height) if foo(Belief)
and [A,B,1.0] then run(speed)
```

Square brackets are used to designate a belief. Each concept and the valence are separated by `,`. Setting valence to 0.0 is ambiguous and will return True if the agent has the belief. The `True` word can be used to designate conditions that will always return true, e.g.:

```
if [True] then talk(belief) call(belief) if [false] or not [TRUE] then
talk(belief) call(belief)
```

Precedence is from left to right or determined by parenthesis and syntax is case sensitive, except for logical operators as beliefs themselves are case-sensitive.

The class has the following main attributes used for mapping rules to actions and maintaining an action queue:

- `rule_list`: is a list of tuples where the first element is the condition, and the second a sequence of actions.
- `belief_token_map`: a dictionary keyed by belief string and valued by the corresponding token (a `concept_pair`, valence tuple)
- `action_token_map`: a dictionary keyed by an action string and valued by a tuple who's first element is action name and following elements the variable names
- `action_name_to_variable_map`: a dictionary keyed by action name and valued by a list of variables that primitive has
- `variable_bindings`: a dictionary keyed by variable name and valued by the value of that variable (defaults to None)
- `action_queue`: a list of action strings to be carried out.
- `current_action`: a pointer to an action in the `action_queue`.

[2]Maybe in the future actions from different conditions can be compartmentalized so that a failure in one condition doesn't affect the others.

2.8 System1Evaluator

This is a helper class for `SocogSystem1Agent`. It handles parsing the rule string and evaluating conditions from primitives supplied in the arguments.

2.9 SocogDASHAction

This class replaces `DASHAction` and implements the agent loops necessary for the socog agent. Currently, `arbitrate_system1_system2` and `bypass_system2` default to choosing system 1 actions if they exist, else it defers to system 2. Override these functions if different behavior is desired. System 1 needs to know when its action is taken, because it keeps a memory of where it is in the sequences of suggested actions. Therefore, whenever system1 actions are chosen the `sys1_action_taken` property should be toggled to `True`.

The class subclasses the `DASHAction` module, so that it has access to the same properties and methods and primitives as that class. Additional primitives are added by default. The `process_belief` primitive takes a belief and calls the belief module's `process_belief` method. The `emit_belief` primitive sets a belief variable and calls the belief module's `emit_belief` method. Lastly, the `belief_conflict` primitive takes a belief variable and checks if the belief conflicts with any beliefs in the agent's belief system. If so it returns a value that evaluates to `True` else it returns a value that evaluates to `False`.

2.10 SocogDASHAgent

This class inherits from the socog classes and calls the respective initializers in the proper order. You can sublass this class to create new agents.

2.11 Tutorial

See tutorial director in Dash2 tutorial section. This is a short tutorial module that can be run with `reddit_hub`. `reddit_hub` contains a set of default comments on a soccer thread that the user agent will parse and convert to beliefs, which it will then assess. It shows how you can subclass the `SocogDASHAgent` and initialize a belief module.

This uses System2 and System1 functions with the `socog_module`. The user is instantiated with a sparse belief system where it believes that Henry and the GreyTeam are the best. Initial forum posts validate this and provide addition connections to the agent which reinforce its beliefs. However, later comments suggest that Henry is Corrupt. After repeated exposure the agent adopts beliefs the reject Henry and GreyTeams Best status and adopt the beliefs that Henry and the GreyTeam are both Corrupt. The rules for system 1 cause the agent to respond with its own beliefs if it reads something that conflicts with those beliefs.

References

[1] Rodriguez N, Bollen J, Ahn Y-Y (2016) Collective Dynamics of Belief Evolution under Cognitive Coherence and Social Conformity. PLoS ONE 11(11): e0165910. https://doi.org/10.1371/journal.pone.0165910