# Model Description

## Overview

Each realization of the simulation is initialized with a new group of legislators. These legislators have a list of priorities and positions associated with a list of issues – one may think of these as the solutions and problems in the Garbage Can Model, with the added dimensionality of priorities. Legislator priorities and positions may or may not conform to ideological party agenda depending on model parameters. A social network connecting the legislators is generated, using both homophily [TBD: reference neded] and preferential attachment [Barabasi and Albert, 1999].

As the simulation runs, bills are sequentially processed as follows:

1. Proposal:
   1. A random legislator is chosen to sponsor a bill.
   2. The sponsor proposes a draft bill on any issue that has not already been addressed by law.
2. Draft circulation among cosponsors:
   1. All legislators connected to the sponsor in the social network are selected as cosponsors.
   2. The cosponsors revise the draft using simulated annealing; new issues may be added to the bill during the revision process and solutions on existing issues may change.
3. Committee review:
   1. The draft is referred to a committee; committees are chosen according to legislators for whom the main issue of the bill is a high priority.
   2. The committee revises the bill by simulated annealing; again, new issues may be added and existing solutions changed as a result.
4. Floor vote:
   1. The bill is referred to the floor for a vote.
   2. If the bill passes by simple majority (> 50% votes), the bill is made into law; i.e., the solutions addressed by the bill are recorded and the issues may not be revisited for the remainder of the realization.

This process is repeated for 200 proposals (or halts if all issues are passed into law).

The model is implented in Python with object-oriented programming, and there are three high-level classes that interact in the model: *State, Legislator,* and *Bill*. These are further described in separate sections below.

The fixed and free parameters of the model are listed and described in Tables TBD and TBD; further implementation detail is addressed in the sections that use them. The simulated annealing parameters are discussed separately in the simulated annealing section of this paper.

|  |  |  |
| --- | --- | --- |
| Parameter | Description | Value |
| Num\_of\_Representatives | Size of the legislative body. | 100 |
| Num\_of\_Issues | Size of the problem set. | 75 |
| Solution\_Bit\_Length | Bit string length of positions and solutions. | 4 |
| Committees\_Per\_Legislator | How many committees a legislator belongs to; sets the lowest priority issue a legislator will serve on committee for. | 4 |
| Satisfaction\_Threshold | The minimum satisfaction a legislator must have with a bill to vote “aye” on it. A calibrated value (see Model Calibration section), but after calibration was fixed for experiment variations. | 0.675 |
| Friend\_Threshold | The minimum homophily (range [-1,1]) for legislators to be considered for preferential attachment. | 0 |
| Minimum\_Friends | The number of links added as each legislator is added to the network. | 5 |

Table 1 - Fixed parameters

|  |  |  |
| --- | --- | --- |
| Parameter | Description | Experiment Variation |
| Unaffilitated\_Fraction | Fraction of the legislative population with no ideological party affiliation. | [0.05, 0.5, 1.0] |
| Green\_Fraction | Fraction of the party-affiliated population belonging to the “Green” party. Remainder belong to the “Yellow” party. | [0.5, 0.75, 1.0] |
| Ideology\_Issues | Ideological platform issues for the parties. | [0, 5] |
| State\_Priorities | High-priority issues for all legislators. | [0, 5] |

Table 2 - Free parameters of the model

## Global Method *binaryTreeDistance()*

The global method *binaryTreeDistance()* is used in various places to calculate homophily between two bit strings – for example, between two legislator positions on an issue, or between a proposed solution and a legislator’s ideal solution (his position on the issue). This function calculates the distance as increasing depth on a binary tree, where the most significant bit (MSB) of the two bit strings is worth 50% similarity, the next MSB is an additional 25 similarity, etc. Two identical strings result in 100% similarity, whereas strings that have only the MSB in common are only 50% similar. At position *i* in the bit strings, a bit’s difference in the binary value results in an additional (½)i+1 reduction in similarity. For an infinitely long bit string length, similarity in all positions but the MSB would result in a 50% similarity as well; practically speaking, however, since our model only uses a bit string length of 4, the maximum achievable *dis*similarity is 93.75%.

This method was used to enable strong issue position correlation between party-affiliated legislators while still allowing for some heterogeneity among their positions, if for example, party positions were only specified to *n* most significant bits of *Solution\_Bit\_Length*. However, this feature was not used in this model (party platform positions are homogeneous among affiliated members) and could be an extension in future work.

Another benefit to this method, however, is that on any given issue, it generates a bimodal, vs. uniform, distribution of all legislator positions. In a completely uncorrelated population (i.e., all independents), a uniform significance of bits towards homophily created an extremely low likelihood of any two legislators having sufficient agreement over the entire solution space to meet any reasonable assumption of friendship threshold. The decreasing significance of the binary tree method is more reasonable in that it could be said to model any two legislators agreeing on the generalities of a solution to an issue (say “for” or “against”) while perhaps differing on implementation details. From a model implementation perspective, it makes the network generation much more tractable using reasonable assumptions of friendship threshold.

## The State Class

The state is a singleton class and the main simulation object. Its main functions are to initialize party agenda (priorities and positions on the issues list), initialize the legislators with party affiliation and issue priorities and positions, call the network generation method (which is actually a static method of the *Legislator* class), and process bills.

### The State: Attributes/Members

Table TBD identifies the *State* class attributes and object members.

|  |  |  |
| --- | --- | --- |
| Attribute | Description | Initialization |
| issues | A list of potential problems to be addressed by the legislative body. | [0, 1 .. Num\_of\_Issues-1] |
| legislators | List of individual legislators making up the legislative body. | (see text) |
| laws | A dictionary, keyed by issue, of solutions to issues that have been passed into law. | {empty} |

Table 3 - The State class attributes and object members

### The State: Initialization

The State initializes a set of state priorities for all legislators, party platform issues and associated seed priorities and positions. State priorities are simply the first n=*State\_Prioritie*s of *State.issues*. Party platform issues are a random sample (n=*Ideology\_Issues*) for each party from among the remaining issues (i.e., not state priorities).

The seed priority list is a concatenation of the platform priorities and the state priorities; this list is rank-ordered so that state priorities are likely to be higher priority than platform priorities. “Seed” refers to the fact that there is a stochastic process (preferential attachment) that generates the actual individual priorities for each legislator, as described in more detail in legislator initialization. To maintain parity of state issue priorities between independents (non-affiliated) and party-affiliated legislators, independents are effectively initialized with their own individual platform of n=Ideology\_Issues priorities; like party priority issues, this is a random sampling for each independent.

For each issue in the seed priority list (platform issues plus state priorities), the associated Green position is a bit string of 1’s of length *Solution\_Bit\_Depth* (4 for our model), and the associated Yellow position is a bit string of 0’s. Green legislators will also have a “1’s” position on Yellow priority issues, and the converse. Thus, our model assumes that although ideological parties may prioritize different problems, they differentiate from each other along non-prioritized problems as well [TBD: reference needed?]. Independents have no *a priori* positions.

Legislators are then generated with a party affiliation, seed priorities list, and a set of positions on issues. Finally, the network is generated.

### The State: Behaviors

Table TBD identifies the methods associated with *State* class behavior. These will be further described in the sections immediately following.

|  |  |
| --- | --- |
| Method | Description |
| step() | Executes one tick of the simulation, wherein a bill is proposed, draft revised with cosponsors and committee, and voted on. |
| getCommitteeMembers() | Returns a list of legislators for whom the bill’s main issue is high priority. |
| circulateBill() | Given a list of reviewers, revises the bill with simulated annealing. |
| putToVote() | Returns the “aye” vote tally on the bill. |
| makeLaw() | Adds any solution addressed in the bill to the list of laws. |

Table 4 - State class methods

#### State Method: step()

The step() method implements steps 1 - 4 as described in the “Model Overview” section above.

#### State Method: getCommitteeMembers()

In this method, any legislator for whom the bill’s main issue is in that legislator’s highest n=*Committees\_Per\_Legislator* priorities will be added to the committee.

#### State Method: circulateBill()

This method takes as argument the list of reviewers (cosponsors or committee) slated to revise a bill and calls the simulated annealer static method *anneal()* with a “rev dash” revision (current solutions in the bill), a modification method *bill.revise()*, and an objective/energy function *bill.measureDisSatisfaction()*. The best solution from the simulated annealing is returned as the revision.

#### State Method: putToVote()

This method calculates the number of legislators for whom the bill’s final solution vector meets the minimum satisfaction threshold.

#### State Method: makeLaw()

If the bill passes, this method creates a new dictionary entry in *State.laws* for each solution in the bill’s solution space.

## The Legislator Class

Legislators propose (sponsor) bills, cosponsor bills, serve on committees, and vote on finalized revisions.

### Legislator: Attributes

Table TBD identifies the attributes of *Legislator* class objects.

|  |  |  |
| --- | --- | --- |
| Attribute | Description | Initialization |
| priorities | A dictionary of priorities accessed by issue. | algorithm (see text) |
| positions | A dictionary of positions accessed by issue. | algorithm (see text) |
| affiliation | Which party (none, green, or yellow) the legislator belongs to. | as determined by the State class |
| links | A list of other legislators this legislator is connected to in the social network. | algorithm (see description of *preferentialHomophilyNetwork())* |

Table 5 - Legislator attributes

### Legislator Initialization

The legislator’s constructor is passed three values from the *State* class: a party affiliation, seed priorities, and default positions.

#### Legislator Priorities Generation

Recall that seed priority issues are generated by the State object and are a rank ordered concatenation of party-prioritized issues (nparty = *Ideology\_Issues*) and state-prioritized issues (nstate = *State\_Priorities*). Starting with element 0 in this priority issues list and proceeding to the end of the list (element N-1, where N = nparty + nstate), a priority 2\*(i + 1) is initially assigned to that issue. Thus, the lowest priority issue will be assigned an initial priority of 2, and the highest priority issue will be assigned an initial value of 2\*N. All remaining issues (*Num\_of\_Issues* – N) are assigned an initial priority of 1.

Over *Num\_of\_Issues*2 iterations, issues are incremented, with the issue to be incremented at each iteration chosen at random according to the probability density of priorities in that iteration. Finally, priorities are normalized to the range [0,1]. This preferential attachment method generates a power law distribution of priorities, so that a legislator places very high priority on a small number of issues but low priority on most issues. The use of the seeded priority issues list means that legislators have some correlation in priorities to the extent that their seed priorities are the same, but heterogeneity is introduced with stochasticism.

#### Legislator Positions Generation

For any party agenda issues passed to the constructor by the state, the legislator adopts the party platform position [TBD discuss assumption, future work]. For all other issues, positions are assigned randomly from the range [0, 2*Solution\_Bit\_Length*-1], inclusive (i.e., a random bit string of length *Solution\_Bit\_Length*).

### Legislator: Behaviors

Table TBD identifies the methods of the *Legislator* class.

|  |  |
| --- | --- |
| Method | Description |
| preferentialHomophilyNetwork() | Generates the social network of the legislative body. See below for detailed description. |
| proposeBill() | Randomly selects an issue to sponsor from among any issues not already passed into law. |
| pickCoSponsors() | Returns all legislators ego is connected to in the social network. |
| getSatisfactionWithBill() | Returns the legislator’s satisfaction with the solutions proposed by the bill. See text below for more detailed description. |

Table 6 - Legislator methods

#### Legislator class static method: preferentialHomophilyNetwork()

This method takes as argument the set of legislators initialized with priorities and positions, and generates a social network among them. Both homophily and preferential attachment are used. Figure TBD is a diagram of the main steps in the network generation.

Find potential friends using homophily

Link to a potential friend using preferential attachment

Remove friend from potential friends

Shuffle legislator list

*ith* legislator

Iterate *Minimum\_Friends* times

Iterate over all legislators

Figure 1 – Diagram of network generation using preferential attachment with homophily.

*Potential friends* are the set of legislator alters who have a minimum similarity of *Friend\_Threshold* total homophily on all issues with ego. Similarity is a sum of homophilies on positions of all issues, weighted by and normalized to ego’s priorities on those issues. Homophily on an issue is the unary inverse (*1 – x)* of the *binaryTreeDistance()* calculated on a pair of legislators’ respective positions on that issue.

Edge assignment is as described in [Barabasi and Albert, 1999]: each time an edge is added, a pdf is generated from the degree distribution in the subnetwork of *potential friends*, and a target node for the edge is selected randomly from that pdf.

#### Legislator object method: proposeBill()

Legislators introduce a bill on an issue randomly selected from any issues not already passed into law. The legislator’s proposed solution for that issue is his position on that issue.

#### Legislator object method: getSatisfactionWithBill()

A legislator’s satisfaction with a bill is calculated the same way as similarity between two legislators: a priority-weighted and -normalized sum of homophilies between the legislator’s issue positions and the bill’s proposed issue solutions.

## The Bill Class

The *Bill* class is a container object for solutions to issues, and has two relevant methods: *revise()* and *measureDisSatisfaction()*.

### Bill: Attributes

Table TBD identifies the attributes of *Bill* class objects.

|  |  |  |
| --- | --- | --- |
| Attribute | Description | Initialization |
| solutions | A dictionary of solutions to issues, indexed by issue | A single entry, as determined by the sponsor |
| main\_issue | The issue on which the bill is introduced | as determined by the sponsor |

Table 7 - Bill attributes

### Bill Initialization

A bill is initialized with a single entry in the solutions dictionary: the solution proposed by the sponsor.

### Bill: Methods

The *Bill* class has only two methods, as required for the simulated annealer: *revise()* and *measureDisSatisfaction()*.

#### Bill class object method: revise()

This method is passed to the simulated annealer as the modifier function. Each time it is called, it makes one change to the bill. An issue is randomly selected from any open issues (not addressed by existing law). If that issue is in the bill’s current solution set, one of the bits in that solution is randomly selected for inversion. If the issue is not already addressed by the bill, a random solution on that issue is added and added to the bills *solutions* dictionary.

#### Bill class object method: measureDisSatisfaction()

This method is passed to the simulated annealer as the objective (or energy) function. Given a set of reviewers (this may be either cosponsors, committee, or, for model metric purposes, the entire legislative body) and calculates the average satisfaction with the bill as determined by a *Legislator* object’s *getSatisfactionWithBill()* method. It returns the negative of this average: higher dissatisfaction with a bill corresponds to higher energy in simulated annealing, lowering the probability of acceptance.

## Simulated Annealing

The simulated annealer implements the Metropolis algorithm for simulated annealing [TBD reference needed]: a state with lower energy than the current state is accepted, while a state with higher energy is accepted with probability . The annealer iterates over the temperatures provided to it in the annealing schedule, and for each temperature, iterates the corresponding number of times provided to it. Each iteration, the modifier function is called on the current state, and the modified state’s energy is calculated per the objective function; the modified state’s energy is then accepted or rejected. The final state returned by the algorithm is the lowest energy (highest satisfaction) one encountered.

For our model, k was chosen to be . This means that, at a temperature of 1.0, a decrease of 0.1 in satisfaction is accepted with probability ½.

The time at temperature annealing schedule was configured linearly, as listed in Table TBD.

|  |  |
| --- | --- |
| Temperature | Time at |
| 1.0 | 2 |
| 0.8 | 4 |
| 0.6 | 6 |
| 0.4 | 8 |
| 0.2 | 10 |

Table 8 - Simulated annealer schedule

# Model Verification, Validation, and Calibration

The model was verified through code review and incremental testing: subunits of functionality were tested by verifying expected intermediate outputs, before implementing more complex functionality.

As an exploratory model, little effort was made to validate outputs. Where applicable, assumptions have been stated and are validated against either literature or reasonable expectation. For example, the *Committees\_per\_Legislator* model parameter is based on the maximum number of committees a congressperson can serve on.

The model parameter *Satisfaction\_Threshold* was calibrated to the value 0.675 to achieve an about 4% passing rate of all initial proposals, with *Unaffiliated\_Fraction* = 0.05, *Green\_Fraction* = 0.5, and *Ideology\_Issues* and *State\_Priorities* each set to 5. A 4% passing rate agrees with typical U.S. Congress pass rates, which are between 2% and 7% [TBD reference?]

# Experimentation Method

A suite of experiments were run against all combinations of parameter values identified in Table 2. An exception was any variation in *Green\_Fraction* when *Unaffiliated\_Fraction* was 1.0, which would have been duplicative results since party affiliation is moot in that case; a separate experiment was run to obtain results with *Unaffiliated\_Fraction* = 1.0 and variations over *Ideology\_Issues* and *State\_Priorities*. To obtain statistically significant results, 30 realizations were simulated for each parameter combination.

Tables TBD and TBD identify the metrics that were calculated or recorded for each proposal during individual run histories and for final aggregate outputs after a completed simulation, respectively. To keep the data set manageable, run histories were not recorded for the main suite of experiments. Instead, histories were only recorded for the baseline model: *Unaffiliated\_Fraction=0.05, Green\_Fraction=0.5, Ideology\_Issues = 10.*

|  |  |
| --- | --- |
| Output | Description/Notes |
| main issue |  |
| congress initial dissatisfaction |  |
| provisional tally | How many votes the initial proposed solution would have received if put immediately to vote. |
| cosponsors |  |
| cosponsor initial dissatisfaction |  |
| cosponsor final dissatisfaction |  |
| committee size |  |
| committee initial dissatisfaction |  |
| committee final dissatisfaction |  |
| number of issues addressed | Number of issues addressed in the final bill version |
| main issue change | binary tree distance between the main issue final solution and the original proposed solution |
| congress final dissatisfaction |  |
| votes |  |

Table 9 - Metrics captured for each proposal during a recorded history

|  |  |
| --- | --- |
| Output | Description/Notes |
| proposals | In rare instances, the legislative body passed laws on all issues in the problem space before the simulation run time. This metric allowed capture of those instances. |
| laws count | How many bills were passed into law. |
| provisions | How many issues were passed into law. |
| total satisfaction | Final legislative body satisfaction with all legislation. |
| total change | average change in the main issues’ positions from proposal to law |
| total votes | How many votes were cast ‘aye’ over the simulation run. |

Table 10 - Aggregate metrics recorded for each simulation realization