Story Graphs

Authorship

This data set contains the details of three different story graphs for the same simple story domain. They were generated by Stephen G. Ware of the Narrative Intelligence Lab (cs.uky.edu/~sgware) at the University of New Orleans in 2018 using supercomputing resources made available by the Louisiana Optical Network Infrastructure, or LONI (loni.org).

This data is released under the Creative Commons Attribution 4.0 International license. In summary, that means you can use, share, and adapt it for your own purposes, but you must credit the original author. A license file with full details is distributed with these files.

Story Domain

The story begins with the protagonist, or player, seeking medicine to heal their sick grandmother. The player's goal is to obtain the medicine and return home. The story ends when the player achieves that goal or dies trying.

This domain has a Medieval setting and includes four discrete locations:

- `Cottage`: The house where the player lives with his or her grandmother.
- `Market`: A market where the merchant sells goods and is protected by the guard.
- `Camp`: The bandit's camp.
- `Crossroads`: A central location connecting the other three locations to one another.

The domain features four characters:

- `Player`: The protagonist.
- `Merchant`: A merchant who is selling medicine and a sword.
- `Guard`: The town guard who protects the market and punishes criminals.
- `Bandit`: A bandit who wants to steal things of value.

The domain features 6 items:

- `Medicine`: A tonic that can heal the player's grandmother, for sale by the merchant.
- `PlayerCoin`: A coin given to the player that can be used to purchase items.
- `BanditCoin`: A coin stashed in a chest in the bandit's camp.
- `GuardSword`: The guard's sword.
- `BanditSword`: The bandit's sword.
- `MerchantSword`: The merchant's sword, which is for sale.

The domain features 1 container:

- `Chest`: A chest in the bandit's camp.

States

A state is defined as an assignment of one value to each the following fluents:

- `alive(<character>) = <boolean>`
  Whether that character is currently alive.
- `armed(<character>) = <boolean>`
  Whether that character has a weapon.
- `criminal(<character>) = <boolean>`
  Whether that character has committed a crime.
- `location(<character>) = <place>`
  That character's current location.
- `location(<item>) = <character OR container>`
  The character who currently has that item or the container it is currently stored in.

The initial state for this story domains is:

- `alive(Bandit) = True`
- `alive(Guard) = True`
- `alive(Merchant) = True`
- `alive(Player) = True`
- `armed(Bandit) = True`
- `armed(Guard) = True`
- `armed(Merchant) = True`
- `armed(Player) = False`
- `criminal(Bandit) = True`
- `criminal(Guard) = False`
- `criminal(Merchant) = False`
- `criminal(Player) = False`
- `location(Bandit) = Camp`
- `location(BanditCoin) = Chest`
- `location(BanditSword) = Bandit`
- `location(Guard) = Market`
- `location(GuardSword) = Guard`
- `location(Medicine) = Merchant`
- `location(Merchant) = Market`
- `location(MerchantSword) = Merchant`
- `location(Player) = Cottage`
- `location(PlayerCoin) = Player`

Beliefs

In addition to what is actually true, states track a limited number of character beliefs:

- `believes(Guard, criminal(Merchant)) = <boolean>`
  Whether the `Guard` believes that the `Merchant` is a criminal.
- `believes(Guard, criminal(Player)) = <boolean>`
  Whether the `Guard` believes that the player is a criminal.
- `believes(Guard, location(Bandit)) = <place>`
  Where the `Guard` believes that the `Bandit` is currently located.
- `believes(Player, location(Bandit)) = <place>`
  Where the `Player` believes that the `Bandit` is currently located.
- `believes(Bandit, location(BanditCoin)) = <character or container>`
  Where the `Bandit` believes the `BanditCoin` is currently located.
- `believes(Merchant, location(BanditCoin)) = <character or container>`
  Where the `Merchant` believes the `BanditCoin` is currently located.
- `believes(Bandit, location(Medicine)) = <character or container>`
  Where the `Bandit` believes the `Medicine` is currently located.
- `believes(Bandit, location(Player)) = <place>`
  Where the `Bandit` believes the `Player` is currently located.
- `believes(Bandit, location(PlayerCoin)) = <character or container>`
  Where the `Bandit` believes the `PlayerCoin` is currently located.
- `believes(Merchant, location(PlayerCoin)) = <character or container>`
  Where the `Merchant` believes the `PlayerCoin` is currently located.

This is the initial state of all character beliefs. Note that `Null` represents "does not know."

- `believes(Guard, criminal(Merchant)) = False`
- `believes(Guard, criminal(Player)) = False`
- `believes(Guard, location(Bandit)) = Null`
- `believes(Player, location(Bandit)) = Null`
- `believes(Bandit, location(BanditCoin)) = Chest`
- `believes(Merchant, location(BanditCoin)) = Null`
- `believes(Bandit, location(Medicine)) = Merchant`
- `believes(Bandit, location(Player)) = Crossroads`
- `believes(Bandit, location(PlayerCoin)) = Player`
- `believes(Merchant, location(PlayerCoin)) = Null`

Goals

An *author goal* represents the designer's constraints on the narrative. For these simple stories, there are two author goals:

- `alive(Player) = False`

- location(Player) = Cottage AND location(Medicine) = Player

The story ends when one of these two author goals is achieved.

The Player's goal is:

- location(Player) = Cottage AND location(Medicine) = Player

The Merchant's goals are:

- location(PlayerCoin) = Merchant AND criminal(Merchant) = False
- location(BanditCoin) = Merchant AND criminal(Merchant) = False
- location(Merchant) = Market

The Guard's goals are:

- alive(Bandit) = False AND criminal(Guard) = False
- If criminal(Player) then alive(Player) = False AND criminal(Guard) = False
- location(Guard) = Market

The Bandit's goal is:

- location(BanditCoin) = Bandit OR location(BanditCoin) = Chest
- location(PlayerCoin) = Bandit
- location(Medicine) = Bandit
- location(Bandit) = Camp

Character goals are listed from highest to lowest priority. For example, the Bandit wants to be at the Camp, but he will leave the Camp if he thinks he can achieve a higher priority goal, such as getting the Medicine.

Actions

A state can change via one of several actions. Every action has preconditions that must be true before it can happen and effects which change the state. Every action also specifies under what conditions an "observing" character sees the action happening. Every action also specifies one or more "consenting" characters who must have a motivation to take the action.

attack(<attacker>, <victim>, <place>)

Preconditions:

- alive(<attacker>) = True
- location(<attacker>) = <place>
- alive(<victim>) = True
- location(<victim>) = <place>
- armed(<attacker>) = True OR armed(<victim>) = False

Effects:

- alive(<victim>) = False
- If criminal(<victim>) = False then criminal(<attacker>) = True

Observing: Any <character> for which location(<character>) = <place>
Consenting: <attacker>

buy(<buyer>, <item>, <coin>, <place>)

Preconditions:

- alive(<buyer>) = True
- location(<buyer>) = <place>
- location(<item>) = Merchant
- location(<coin>) = <buyer>
- location(Merchant) = <place>

Effects:

- location(<item>) = <buyer>
- location(<coin>) = Merchant

Observing: Any <character> for which location(<character>) = <place>
Consenting: <buyer> and Merchant

loot(<looter>, <item>, <victim>, <place>)

Preconditions:

- alive(<looter>) = True
- location(<looter>) = <place>
- location(<item>) = <victim>
- alive(<victim>) = False
- location(<victim>) = <place>

Effects:

- location(<item>) = <looter>

Observing: Any <character> for which location(<character>) = <place>
Consenting: <looter>

report(<reporter>, <bandit_place>, <reporter_place>)

Preconditions:

- alive(<reporter>) = True
- location(<reporter>) = <reporter_place>
- alive(Guard) = True
- location(Guard) = <reporter_place>
- believes(<reporter>, location(Bandit)) = <bandit_place>

Effects:

- believes(Guard, location(Bandit)) = <bandit_place>

Observing: Any <character> for which location(<character>) = <reporter_place>
Consenting: <reporter>

rob(<robber>, <item>, <victim>, <place>)

Preconditions:

- alive(<robber>) = True
- location(<robber>) = <place>
- armed(<robber>) = True
- location(<item>) = <victim>
- alive(<victim>) = True
- location(<victim>) = <place>
- armed(<victim>) = False

Effects:

- location(<item>) = <robber>
- If criminal(<victim>) = False then criminal(<robber>) = True

Observing: Any <character> for which location(<character>) = <place>

Consenting: <robber>

take-out(<taker>, <item>, Chest, Camp)

Preconditions:

- alive(<taker>) = True
- location(<taker>) = Camp
- location(<item>) = Chest

Effects:

- location(<item>) = <taker>

Observing: Any <character> for which location(<character>) = Camp

Consenting: <taker>

walk(<walker>, <from>, <to>)

Preconditions:

- alive(<walker>) = True
- location(<walker>) = <from>

Effect:

- location(<walker>) = <to>

Observers: Any <character> for which location(<character>) = <from> OR location(<character>) = <to>

Consenting: <walker>

Axioms

There are also several axioms which are applied after an action occurs to update the state. These are not optional and must be applied if they can be. Axioms only have preconditions and effects. Everyone observes axioms, and nobody needs to consent for them to happen.

armed(<character>)

Preconditions:

- location(GuardSword) = <character> OR
  location(BanditSword) = <character> OR
  location(MerchantSword) = <character>
- armed(<character>) = False

Effect:

- armed(<character>) = True

unarmed(<character>)

Preconditions:

- location(GuardSword) != <character>
- location(BanditSword) != <character>
- location(MerchantSword) != <character>
- armed(<character>) = True

Effect:

- armed(<character>) = False

wants-justice(Guard, <character>)

Preconditions:

- alive(Guard) = True
- believes(Guard, criminal(<character>)) = True

Effect:

- criminal(<character>) = True

see-character-at(<observer>, <target>, <place>)

Preconditions:

- alive(<observer>) = True
- location(<observer>) = <place>
- location(<target>) = <place>
- believes(<observer>, location(<target>)) != <place>

Effect:

- believes(<observer>, location(<target>)) = <place>

see-character-not-at(<observer>, <target>, <place>)

Preconditions:

- alive(<observer>) = True
- location(<observer>) = <place>
- location(<target>) != <place>
- believes(<observer>, location(<target>)) = <place>

Effect:

- believes(<observer>, location(<target>)) = Null

see-item-on(<observer>, <item>, <character>, <place>)

Preconditions:

- alive(<observer>) = True
- location(<observer>) = <place>
- location(<item>) = <character>
- believes(<observer>, location(<item>)) != <character>
- location(<character>) = <place>

Effect:

- believes(<observer>, location(<item>)) = <character>

see-item-not-on(<observer>, <item>, <character>, <place>)

Preconditions:

- alive(<observer>) = True
- location(<observer>) = <place>
- location(<item>) != <character>
- believes(<observer>, location(<item>)) = <character>
- location(<character>) = <place>

Effect:

- believes(<observer>, location(<item>)) = Null

see-item-in(<observer>, <item>, Chest, Camp)

Preconditions:

- alive(<observer>) = True
- location(<observer>) = Camp
- location(<item>) = Chest
- believes(<observer>, location(<item>)) != Chest

Effect:

- believes(<observer>, location(<item>)) = Chest

see-item-not-in(<observer>, <item>, Chest, Camp)

Preconditions:

- alive(<observer>) = True
- location(<observer>) = Camp
- location(<item>) != Chest
- believes(<observer>, location(<item>)) = Chest

Effect:

- believes(<observer>, location(<item>)) = Null

About the Story Graphs

A story graph is a directed, labeled graph with two kinds of edges. A node n in the graph represents a state, a unique configuration of people, places, and items in the story world.

A temporal edge $n_1 \overset{a}{\rightarrow} n_2$ may exist from node n_1 to node n_2 for action a if the preconditions of a are satisfied in n_1 and n_2 is the state that would result from applying the effects of a to n_1 and then applying any relevant axioms. When action a requires `Player`'s consent, that edge is called a *player action*. When action a requires any other (non-player) character's consent, that edge is called an *NPC action*. An action can be both a player action and an NPC action if it requires both player and non-player consent (e.g. `buy(Player, Medicine, PlayerCoin, Market)` requires the consent of both `Player` and `Merchant`); these are called *mixed actions*. Actions requiring only the player's consent are *player only actions*, while those requiring only the consent of NPCs are *NPC only actions*.

An epistemic edge $n_1 \overset{c}{\rightarrow} n_2$ may exist from node n_1 to node n_2 via character c if, when the world is in state n_1, character c believes the world to be in state n_2. When following an epistemic edge, character c's beliefs become true. For example, if the following propositions are true in n_1:

- `location(Bandit) = Camp`
- `believes(Guard, location(Bandit)) = Crossroads`
- `believes(Player, location(Bandit)) = Market`

And c is the Guard, then the following propositions will be true in n_2:

- `location(Bandit) = Crossroads`
- `believes(Guard, location(Bandit)) = Crossroads`
- `believes(Player, location(Bandit)) = Crossroads`

Note that no second order beliefs (beliefs about beliefs) are tracked. In the above example, the Guard does not reason about where he thinks the player thinks the bandit is—or rather, the Guard always assumes the player believes the bandit is wherever the guard thinks the bandit is.

There are three story graphs in this data set. For every state in all three story graphs, there always exists a temporal edge for every player only action whose preconditions are satisfied in that state. In other words, if we consider a story graph as a map of an interactive narrative, then it is always possible for a player to take any action whose preconditions are satisfied. Different story graphs have different policies about mixed edges.

The `Full` Story Graph

The `full` story graph represents a large space of many possible narratives that could take place in this story domain. NPC actions exist in this story graph when the action can be explained for all the consenting NPCs. An action is *explained* for some character c when c believes the action will causally contribute to a plan that achieves one of c's goals, where that plan is 3 or fewer steps long. When in state n and checking whether action a can be explained for character c, one first follows the epistemic edge from n for c (if it exists) and then searches for a temporal path of 3 or fewer edges that ends in a state where one of c's goals is achieved. In short, this means that an action is explained when all the characters who take that action think it will help them achieve one of their goals. For full details, see the following paper:

> Alireza Shirvani, Rachelyn Farrell, Stephen G. Ware. "Combining Intentionality and Belief: Revisiting Believable Character Plans." In *Proceedings of the 14th AAAI International Conference on Artificial Intelligence and Interactive Digital Entertainment*, pp. 222-228, 2018.

The `full` graph contains mixed actions only when they can be explained for all the NPCs who must consent (e.g. the `Merchant` will only sell something to the `Player` when the `Merchant` expects that action to contribute to achieving one of her goals).

Note that, even though NPCs only form plans of length 3, this often leads to plans that seem longer. For example, in the initial state, the `Bandit` is at the `Camp` and believes the `Player` is at the `Crossroads` (though the `Player` is actually at the `Cottage`). The `Bandit` can form a 2 step plan to get the `PlayerCoin`: first walk to the `Crossroads` and then rob the `Player`. The `Bandit` cannot form a 3 or fewer step plan to get the `Medicine`. However, as soon as the `Bandit` walks to the `Crossroads` and discovers that the `Player` is not there, the `Bandit` can now form a 3 step plan to get the `Medicine`: walk to the `Market`, kill the `Merchant`, loot the `Medicine`.

The `Pruned` Story Graph

The `pruned` story graph represents a specific interactive story which was generated by starting with the `full` story graph and then intelligently removing edges. Details and examples are described in the following paper:

> Stephen G. Ware, Edward T. Garcia, Alireza Shirvani, Rachelyn Farrell. "Multi-Agent Narrative Experience Management as Story Graph Pruning." In *Proceedings of the 15th AAAI International Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2019.

For simplicity, we also include a short description of how the graph was pruned below. Edges were removed according to this algorithm:

```
For each of the pruning criteria below (in order),
    For each node in the graph,
        For each edge out from that node, if it meets the criteria, prune it.
```

The pruning criteria are:

1. **Shorter Plan Pruning:** Given two NPC actions for the same NPC, if both actions can be explained by a plan that achieves the same goal, but one plan is shorter than the other, we prune the action that is the start of the longer plan.
2. **Lazy NPC Pruning:** Given an action by an NPC that can be explained by a plan to achieve a goal, if there also exists a player action that the NPC believes can be explained by a plan to achieve the same goal, the NPC action is pruned. In other words, if an NPC has a way to achieve their goal, but thinks the player is also working to achieve that goal, the NPC will not act.

3. **Unique Ending Pruning:** Given two actions by the same NPC, we remove the one which most reduces the number of endings that are possible. Recall that there are two possible endings: the player brings the medicine home or the player dies. This pruning will never remove an NPC's last action. It is more important that an NPC be seen to follow through with their plans, even if it means reducing the number of possible endings.
4. **Goal Priority Pruning:** If an NPC has two actions, one that achieves goal g_1 and another that achieves goal g_2, prune the one that is explained by the lower priority goal. In the earlier list of character goals, goals are listed from highest to lowest priority.
5. **Cycle Pruning:** If an NPC has more than one action they can take in a state, and one action is part of a cycle of 3 or fewer edges, remove the edge that is part of a cycle. If every edge in a cycle is that NPC's only action, remove the edge that is part of the longest plan (i.e. it is better to interrupt a plan with 3 more steps left than a plan with only 2 more steps left).
6. **Arbitrary Pruning:** If an NPC has more than one action left, choose one arbitrarily and prune the rest. Also, remove all epistemic edges and all outgoing action edges from terminal states.
7. **Dead End Pruning:** Given an edge $n_1 \xrightarrow{a} n_2$, where a terminal state is reachable from n_1, a is an NPC action, and a terminal state is not reachable from n_2, remove the edge. This ensures that the story can always be finished without removing any player only edges.

The `Random` Story Graph

The `random` story graph represents a randomly generated story. This graph does not reason about any character beliefs; it only reasons about the observable fluents and has no epistemic edges. In every state, all possible player action edges are included. In any given state, there is a 75% chance that a random non-player character action will be included. However, the random NPC action must not make it such that this plan would no longer be possible: The `Player` walks to the `Crossroads`. The `Player` walks to the `Market`. The `Player` buys the `Medicine` from the `Merchant`. The `Player` walks to the `Crossroads`. The `Player` walks to the `Cottage`. This restriction is to ensure that there is at least one feasible way to achieve both endings from the initial state. Finally, after generating this graph, dead end pruning was applied as for the previous graph. This is to ensure that the story can always be finished.

File Format

In general, there are two kinds of files: label and structure files. Label files are in `.txt` format, and define strings of text, one per line. The first line is line 0, the next is line 1, etc. Structure files are in `.csv` format and refer to labels using the line numbers of those labels. All files should have Unix-style line endings. All of the file names given below exist for all three graphs, prefixed by the graph's directory name (though some may be empty). For example, the list of fluents for the Full Story Graph is in the `full` directory and named `full_fluents.txt`; the equivalent list for the Pruned Story Graph is in the `pruned` directory and named `pruned_fluents.txt`, etc.

<graph>_fluents.txt

This file labels all the fluents for the graph. A fluent is an element of a state which can change. For example, line 0 of `pruned_fluents.txt` should be `alive(Bandit)`, which is `True` in states where the `Bandit` is alive and `False` in states where the `Bandit` is dead. The random graph does not reason about beliefs, so it has fewer fluents than the full and pruned graphs.

<graph>_values.txt

This file labels all the values that fluents can have in the graph. Values include `True`, `False`, `Null`, places, characters, items, etc. For example, line 0 of `pruned_fluents.txt` should be `Bandit`, which refers to the character who wants to steal items of value.

<graph>_states.csv

This file defines the values of each fluent for each state node. Each line is a new state, with the state for node 0 defined on the first line, the state for node 1 defined on the next, etc. Node 0 is the initial state of the story. Each line has a comma-separated list of values, one per fluent, in order. Because `pruned_fluents.txt` has 32 lines, each line of `pruned_states.csv` has 32 values. The first fluent for the pruned graph (line 0 of `pruned_fluents.txt`) is `alive(Bandit)`. The first value on the first line of `pruned_states.csv` is 11, so this can be read as "In state 0, fluent 0 has value 11." Line 11 of `pruned_values.txt` is `True`. Thus, in node 0, `alive(Bandit)=True`; in other words, the Bandit is alive in the initial state of the story.

<graph>_actions.txt

This file labels all the actions that occur in the graph (i.e. all possible labels for temporal edges). For example, line 0 of `pruned_actions.txt` should be `attack(Bandit, Guard, Crossroads)`, which refers to the action where the `Bandit` attacks and kills the `Guard` while both characters are at the `Crossroads`.

<graph>_temporal.csv

This file defines all the temporal edges in a graph. The first line defines temporal edge 0, the next defines temporal edge 1, etc. Each edge has three values separated by commas. The first value, called the "tail," is the number of a state in which the action's preconditions are met. The second value is the number of the action. The third value, called the "head," is the number of the state that results from taking the action. For example, the first line of `pruned_temporal.csv` is `0,277,1`. This means that, in state 0, the preconditions of the 277^{th} action, which is `walk(Player, Cottage, Crossroads)`, are met, and if that action occurs, the story will now be in state 1.

<graph>_agents.txt

This file labels all the agents that have beliefs in the graph (i.e. all possible labels for epistemic edges). For example, line 0 of `pruned_agents.txt` should be `Bandit`. The random graph does not reason about beliefs, and thus has no epistemic edges, so `random_agents.txt` is empty.

<graph>_epistemic.csv

This file defines all the epistemic edges in the graph. The first line defines epistemic edge 0, the next defines epistemic edge 1, etc. Each edge has three values separated by commas. The first value, called the "tail," is the number of a state in which an agent has beliefs. The second value is the number of the agent. The third value, called the "head," is the state which describes the agent's beliefs. For example, the first line of `full_epistemic.csv` is `0,0,1`. This means that, in state 0, agent 0 believes the state to be state 1. According to `full_agents.txt`, agent 0 is the `Bandit`. There are several differences between state 0 and state 1 in `full_states.csv`. For example, in state 0, fluent 27 has value 3, but in state 1 fluent 27 has value 4. In other words, in the initial state of the story, `location(Player)=Cottage`, but the `Bandit` believes that `location(Player)=Crossroads`. Epistemic edges are only defined when an agent's beliefs differ from the actual state. When no epistemic edges exist for an agent, we assume their beliefs are identical to the actual state. Epistemic edges have been removed from the pruned graph, and no belief reasoning was done for the random graph, so `pruned_epistemic.csv` and `random_epistemic.csv` are empty.

<graph>_goals.txt

This file labels all the goals that agents can work toward in the graph. For example, line 0 of `pruned_goals.txt` should be:

```
intends(Bandit, (location(BanditCoin) = Bandit | location(BanditCoin) = Chest))
```

This is the `Bandit`'s goal that he should either be holding his coin or that it should be in his chest. Because NPC actions were chosen at random for the random graph, no goal reasoning was performed, so `random_goals.txt` is empty.

<graph>_plans.csv

This file defines plans that explain why agents would take certain actions. Recall that an NPC action will only appear in the full graph if the agent believes it will contribute to achieving one of its goals. The format for a plan is a sequence of comma-separated values:

- The number of the temporal edge with which this plan is associated
- The number of the agent for whom this plan achieves a goal
- The number of the goal this plan is meant to achieve
- One or more numbers of actions that must be executed next, in order, to achieve that goal

For example, line 0 of `pruned_plans.csv` should be `1,0,2,241`. This means it is associated with temporal edge 1 (which is defined on the second line of `pruned_temporal.csv` and is the temporal edge going from state 0 via action `walk(Bandit, Camp, Crossroads)` to state 2). The agent associated with this plan is agent 0, the `Bandit`. The goal the agent is trying to achieve is goal 2, `intends(Bandit, (location(PlayerCoin) = Bandit | location(PlayerCoin) = Chest))`. There is one other action in the plan, action 241, which is `rob(Bandit, PlayerCoin, Player, Crossroads)`. In other words, if we ask, "In state 0, why would the `Bandit` consent to walk to the `Crossroads`?" this plan answers with, "Because the `Bandit` believes he can then rob the `Player` of the `PlayerCoin`." Note that this plan will not actually work, because the player is not at the `Crossroads`. However, the `Bandit` believes it will work, because the `Bandit` believes the `Player` is at the `Crossroads`, so it is enough to explain why the `Bandit` would walk to the `Crossroads`.