Large Language Models as Narrative Planning Search Guides

Rachelyn Farrell and Stephen G. Ware
University of Kentucky

***Abstract*—Symbolic planning algorithms and large language models have different strengths and weaknesses for story generation, suggesting hybrid models might leverage advantages from both. Others have proposed using a language model in combination with a partial order planning style algorithm to avoid the need for a hand-written symbolic domain of actions, or generating these domains from natural language input. This paper offers a complementary approach. We propose to use a state space planning algorithm to plan coherent multi-agent stories using hand-written symbolic domains, but with a language model acting as a guide to estimate which events are worth exploring first. We present an initial evaluation of this approach on a set of benchmark narrative planning problems.**

***Index Terms*—narrative planning, story generation, language models**

I. Introduction and Related Work

Interactive narratives play an important role in many virtual environments for entertainment and education. Ensuring the narrative remains believable and well-structured as it adapts to player input is a challenge that artificial intelligence and machine learning techniques can help to address.

Early interactive narrative systems, like *Tale-Spin* [1], *Universe* [2], and *Façade* [3], used symbolic models of their story domains. Characters, places, objects, and states are typically represented in some form of predicate logic. Events have preconditions which must be true before they can happen and effects which change the narrative state. These systems are primarily reactive, meaning they choose and commit to a single action based on the narrative so far with minimal look-ahead. The reactive approach scales well, and continues to be a popular choice for modern interactive narratives like *Prom Week* [4], *Versu* [5], *Why Are We Like This?* [6], and many others.

There has also been considerable research on using AI planning to ensure the structure of interactive stories. Plans and planning offer a rich, formal, generative, symbolic model of storytelling [7] that also uses predicate logic and events with preconditions and effects. Planning algorithms have been used to create many interactive narratives [8]–[11]. There has also been considerable research on augmenting planning algorithms with models of story structure [12]–[14]. Planning-based systems differ from reactive systems by performing an extensive backtracking search over the space of all possible stories. This grants them a greater ability to ensure story quality and structure, but it comes at the cost of scalability. The space of all possible stories in a domain quickly becomes too large to explore efficiently, limiting the applicability of planning to large story worlds and prompting research into fast narrative planning [15], [16].

With the recent explosion of research on deep learning and large neural language models (LLMs), there have also been many applications of language generators to storytelling. One example among many is *AI Dungeon* [17], which mediates the player's natural language input to GPT-2 [18] to mimic role-playing games. LLMs are generally used reactively, in the sense that they do little or no search through the space of possible stories, but generate and commit to the next word of text based on the current context. While training LLMs is expensive, text generation is relatively cheap. Unlike symbolic reactive storytelling systems, LLMs can draw from a large training set of characters, places, objects, and events without requiring the user to define them by hand in advance. However, while generated text can be coherent within a sliding window, it is difficult to maintain global coherence or to generate a goal-directed story with a language model alone. Several researchers have worked to address these limitations with respect to story generation [19]–[22].

One interesting method of achieving global story coherence was proposed by Ye et al. [23] It mimics a partial order planning algorithm [24] that works backwards from the end of the story. For the story goal, and for each event, the system determines the preconditions that must be met. For each precondition, it finds an existing event in the story or adds a new event to the story which has an effect that satisfies that precondition. The resulting story is a causal network of events that improves coherence [25]. Its key distinction from a traditional partial order planner is that, rather than relying on a hand-authored symbolic domain of events, the system prompts an LLM to determine preconditions and to generate events which can satisfy them. Another method was proposed by Kelly et al. [26] It uses an LLM to generate symbolic story domains based on input text from example stories. The generated domains are then solved using the Glaive narrative planner to produce new stories, which are then converted into natural language with the LLM. These neurosymbolic story generation approaches both combine the coherence of a planner with the on-demand content generation of an LLM.

However, content generation is a double-edged sword. The ability of LLMs to introduce new story elements can be an advantage or a liability depending on the application. When stories are told in text, as with *AI Dungeon*, there is little cost to introducing new elements. However, many interactive narratives take place in 3D virtual game-like environments where art assets, animations, and dialog are fixed. The ability of an LLM to introduce a dragon into a 3D game's story, however fitting it might be, is useless if the game does not have a rigged and animated dragon model. There is preliminary evidence that deep learning might eventually generate these art assets on demand as well [27], but this technology is in its infancy and raises significant ethical questions about attribution [28]. Regardless, some authors might want to limit a storytelling system to a fixed set of assets for aesthetic purposes, and commercial virtual environments may impose these limits to ease quality assurance.

We are interested in an approach that maintains the use of a human-authored symbolic domain and the story quality assurances afforded by narrative planners like Sabre [29] which conduct extensive backtracking search to guarantee action explainability. In line with the LLM-Modulo framework [30], which suggests LLMs can be useful as idea generators to aid in planning and reasoning tasks, we aim to leverage the general knowledge and storytelling capabilities of an LLM to guide the planner toward solutions more efficiently. In particular we are concerned with the space explosion introduced by Sabre's belief model, which tracks arbitrarily nested theory of mind in order to improve character believability. In this work we explore whether the latent knowledge embedded in LLMs can effectively serve as a believability estimate, something that existing search heuristics lack.

II. Background

In forward state-space planning, actions are added onto the ends of plans until a goal state is reached. Classically, a plan that reaches a goal state is called a solution. Narrative planners however may impose additional constraints on solutions to ensure that they meet certain desired narrative criteria, such as for characters to act believably. Toward this end, a planner can define *characters* as special kinds of entities that have goals and can only participate in actions that contribute to achieving their goals. We use the Sabre narrative planner, which additionally models character beliefs, and ensures that the plans or *explanations* used to justify a character's actions are consistent with their beliefs under limited observability.

For Sabre, a solution is a plan that 1) reaches a goal state and 2) contains only actions that are *explained*. For a full definition of action explainability we refer the reader to the Sabre paper, but to summarize: An action is explained in general when it is explained for each of its consenting characters—the characters who are willing participants in the action. An action is explained for a character when it is the next step in some plan that achieves that character's goals, according to their beliefs. The existence of such a plan indicates that the character will appear to be taking the action for a reason, thus the action is explained for them. The planner tracks this information during search by identifying when character goals are achieved and propagating explanations backward, marking that character's prior actions as explained if they contributed to the achievement of the goal in a way that is consistent with the character's beliefs.

To plan efficiently in large spaces, planners need to consider actions in an intelligent order that prioritizes those that are likely to lead to a solution. Search algorithms like A* can solve classical planning problems efficiently by accurately estimating how close a given state is to a goal state. However, for Sabre, reaching the goal state is not sufficient to qualify a plan as a solution; additional work must be done to ensure that the actions are believable for the characters. It is this additional work that classical planners fail to estimate; i.e. the likelihood that a plan will become explained. A good Sabre heuristic needs to be able to estimate whether characters are acting believably. The Glaive heuristic [15] does this for Glaive, by estimating distance to character goals in addition to the story goal. However, it assumes full observability, since Glaive does not model character beliefs, and therefore does not perform well when Sabre's limited observability constraints are applied.

Sabre's belief model significantly increases the size of the search space because the planner needs to consider not only the actions that are possible, but also those that characters believe to be possible, since this might explain their actions. While we ultimately envision an LLM-based heuristic used in combination with an intelligent search algorithm, at this stage it is not clear what the ideal search algorithm for Sabre would even be [31], nor how best to handle theory of mind questions with LLMs. To begin this line of research, we evaluate whether story suggestions made by an LLM tend to reflect believable behavior according to Sabre's model, and discuss how domain modeling and prompt text decisions can affect this.

We present a search algorithm that explores the space of a narrative planning problem by eliciting suggestions from an LLM for what should happen next in each different scenario, and attempting to follow this guidance. This algorithm is meant to be somewhat thorough, rather than efficient, so that we can collect data from a large part of the problem space and analyze it according to character believability. Like a classical planner, but unlike Sabre, this method considers only the actions that are actually possible from the current state. We believe it is valuable to study how the LLM behaves in this part of the space before we approach the added complication of epistemic nodes and theory of mind (though we discuss our plans for this in Section V).

Problem Definition

We begin by summarizing the formal definitions of our planning problem. A Sabre problem [29] is defined as $\langle C, F, A, T, U, s_0 \rangle$, where:

- C is a set of *characters*: special entities which should appear to have beliefs and intentions.
- F is a set of state fluents: Boolean, nominal, or numeric properties whose values can change over time. Sabre defines a host of logical literals and operations over these fluents, including nested epistemic fluents representing complex character beliefs.
- A is a set of *actions*: STRIPS-like [32] operators that specify preconditions for when they can occur and effects that change the world state afterward. Sabre actions also define a set of *consenting characters*—those who are participating in the action intentionally and must have a

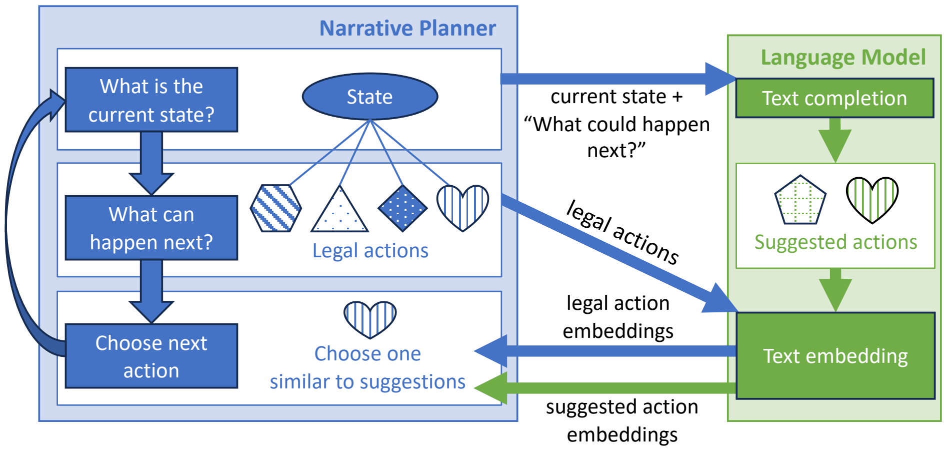

Fig. 1. The narrative planner compares available actions based on suggestions generated with a large language model. Source code for this is available on GitHub (https://github.com/rac7hel/planning-stories-neurally).

reason for doing so—as well as an *observation function* to determine which characters are aware of the action when it occurs.

- T is a set of *triggers*, which are like actions except that they happen automatically. Whenever their preconditions are satisfied in a state, their effects apply immediately. Triggers do not have consenting characters or an observation function.
- U is a set of *utility functions*: expressions that define preferences over states. This includes the *author utility function*, which represents the states the planner should attempt to reach (the story goal), and one utility function for each character to represent their goals.
- s_0 is the *initial state*, an assignment of a starting value for every fluent in F and any wrong beliefs that characters initially hold.

Given a problem so defined, the planner's task is to find a plan, or sequence of actions, that transforms the world from the initial state into one in which the author utility is improved (the story goal achieved) and all actions are explained for their consenting characters.

III. Method

In this section we define the search method used for our study, a simple uniform-cost search algorithm with an LLM-based cost function. The process is illustrated in Figure 1. At each step in the search, the planner (Sabre) describes the current story context to an LLM (*gpt-3.5-turbo*, May 2023), prompting it to suggest what happens next. The planner then prioritizes its available actions based on their similarity to the suggestions, where similarity is determined by comparing the LLM embeddings of the legal and suggested actions.

A. Story Context Description

Each node visited by Sabre corresponds to some unique story in the problem space (a sequence of actions to be taken from the initial state of the problem). At each step during search, our augmented planner begins by creating a natural language description of the current context using the following information (see Appendix for the full text of this example):

1) The initial state: "The hero is alive. The merchant is alive. The guard is alive..."
2) Each character's goals: "The hero wants to bring the medicine to the cottage. The bandit wants to collect valuable items..."
3) The names of the actions in the domain: "Characters can walk, buy, loot, attack, rob, report, and take."
4) The story goal: "The story must end with the hero either being attacked or having the medicine at the cottage."
5) The history (actions that have already happened, if any): "The story begins with the following steps: The merchant walks from the market to the crossroads." (The example is for the first step in the story, so history is empty.)

We created handwritten grammars for each of the Sabre benchmark domains because we wanted to ensure that these translations accurately reflect the domain model for this experiment. In practice this step could also be automated.

B. Prompt for Suggestions

The story context description is then combined with a prompt to suggest followup actions. There are other approaches we could have taken to elicit guidance for the planner from the LLM. For example, we could have asked it to rank all the available actions, or to produce a numeric estimate for the goodness or believability of an action. These would produce output that the planner could more readily use, but it would be much less reliable because LLMs are not equipped to handle complex reasoning. Our method leaves the reasoning to the planner and avoids over-burdening the LLM, asking it to do only the task it was designed for, namely predicting what comes next.

That said, there is a slight complication we have to account for. If we only ask the LLM for one suggestion at a time (e.g. "What should happen next?"), then it will prefer a specific story arc based on its training, usually one that favors the story's protagonist. If the planner always veers toward favorable endings, it can fail to explore critical parts of the space (e.g. the outcomes that explain the antagonist's actions). Asking the LLM for two different suggestions forces it to produce an alternative to its favored response, and allows the planner to pursue multiple branches of the space.

We use the prompt text:

List n different actions that could happen next.
State each as a short sentence on its own line.

where $n > 1$; by default $n = 2$. We also tested $n = \{3, 4\}$ in three example problems (Grandma Lose, Secret Agent, and Raiders) and did not observe any significant difference in performance.

We used OpenAI's *gpt-3.5-turbo* (May 2023) for all text completion tasks, limiting the response to 25 tokens (when $n = 2$), and using a temperature of 0.3. We did not fine-tune the model, but since our context description ends with a list of prior actions in the format we hope to generate, this is essentially few-shot prompting. The only exception is the root node, which represents the empty plan, where there are no prior events in the story.

C. Cost Function

The response from the LLM is parsed into n suggestion strings. For Step 1 of the Grandma Lose example (Appendix), the two suggestions are "The hero walks to the market" and "The merchant buys the medicine". At this point the planner identifies all the available actions, translates them into natural language sentences using the grammars discussed earlier, and obtains LLM embedding vectors for all sentences. The GPT model at the time of this study did not provide access to text embeddings; we used OpenAI's *text-embedding-ada-002* model (May 2023) for all embedding requests.

The text embeddings of the legal actions are then compared with those of the suggested actions. Actions that are good matches to any of the LLM's suggestions are the ones we want the planner to visit first. Therefore lower costs should be assigned to actions that are similar to any suggestion, and higher costs to those that are not. Let $a \in A$ be one of the grounded actions in the domain; and let S be a set of suggestion strings from the LLM output. We define the distance between a and its closest match in S as:

$$d(a, S) \leftarrow \min_{s \in S} \left(1 - \frac{1 + \textsc{CosSim}(\textsc{Em}(a.\textsc{nl}), \textsc{Em}(s))}{2} \right)$$

where $a.\textsc{nl}$ is the natural language translation of the action a (discussed in Section III-A) and the function $\textsc{em}()$ returns the text embedding of the given string. The cosine similarity $\textsc{CosSim}()$ between each pair of embeddings is inverted and scaled between 0 and 1 to avoid negative costs, and the smallest distance value is returned. During search (Algorithm 1), actions with smaller distances are assigned lower step costs, causing the planner to prioritize them.

We use a constant scaling factor z to preserve some number of decimal places when rounding these cosine distance values to assign action costs. By default we preserve two decimal places ($z = 100$). Consider an example: The set of distance values between each possible action and each suggestion, for one step of an example search, range from 0.032 to 0.108. Preserving two decimal places means we assign costs ranging from 3 to 11. Preserving only one ($z = 10$) would reduce precision unhelpfully (costs could only be 0 or 1). Preserving three ($z = 1000$) would yield a much wider range of costs (32...108) and thus a more precise ordering. When we compared $z = \{100, 1000\}$ in three problems (Grandma Lose, Secret Agent, and Raiders), we did not observe a significant difference between the number of nodes visited or the percent of nodes that were explained.

Continuing with the example in the Appendix, there are 12 possible actions, including some that are explained and necessary for most solutions (e.g. The hero walks from the cottage to the crossroads), some that would lead to an ending but are never going to be explained (e.g. The hero attacks himself), some that can be explained but are irrelevant for the story goal (e.g. The merchant reports seeing the bandit at the camp to the guard) and many in between.

As it happens, neither of the suggested actions is possible at this moment: The hero cannot walk directly to the market because it is only connected to his cottage via the crossroads, so he would have to walk there first; and the merchant cannot buy the medicine because she is the merchant---she can't buy from herself! As this example illustrates, the LLM often gives poor suggestions for the very first call, when the history is empty. Having at least one action in the history helps by prompting it with the expected format for actions and some immediately relevant context.

Since none of the possible actions are a perfect match, none will be given a cost of 0. In this case, the lowest cost is 4, which is appropriately assigned to the very important action, "The hero walks from the cottage to the crossroads", due to its similarity to the suggestion "The hero walks to the market". Another action which is also helpful (it is explained and leads to some solutions) is tied for the same cost: "The merchant walks from the market to the crossroads", based on its similarity to "The merchant buys the medicine". So even though the LLM's suggestions are not applicable in this state, the planner does its best to follow them (the hero walks somewhere, the merchant does something). In this case, the unhelpful actions mentioned above are assigned higher costs (8 and 9) and will not be considered until much later in the search, if at all.

D. Search

We now define the GPT-guided uniform cost search algorithm, which we label UCS_GPT. Uniform cost search (UCS) always visits the plan with the lowest cumulative action cost. Breadth-first search is an example of UCS in which all action costs are 1, so the cost of a plan is always equal to the plan's length. In UCS_GPT, the cost of a plan is the sum of the GPT-based costs of the actions in the plan. Thus, longer plans which adhere closely to the suggestions can be visited sooner than short plans which do not.

The search process is formalized in Algorithm 1. Before search begins, the planner grounds and simplifies the problem.

Algorithm 1 GPT-Guided Uniform Cost Search

```
1: Let MAX_DEPTH be a depth limit, $A$ the set of domain actions, and $z$ a precision constant (default 100).
2: STATE($\pi$) denotes the state reached by executing the plan $\pi$ starting from the initial state of $p$.
3: function UCS_GPT(problem $p$)    ▷ Returns a solution, or failure
4:     $\pi \leftarrow$ an empty plan (sequence of actions)
5:     $q \leftarrow$ an empty min-priority queue of plans
6:     $q$.PUSH($\pi$, cost $= 0$)
7:     loop
8:         $\pi \leftarrow q$.POP()
9:         if STATE($\pi$) $\models p$.goal and $\pi$ is explained then
10:            return $\pi$
11:        else if LENGTH($\pi$) $+ 1 \leq$ MAX_DEPTH then
12:            $S \leftarrow$ PROMPTLLM($p, \pi$)
13:            for all $a \in A$ do
14:                if STATE($\pi$) $\models a$.precondition then
15:                    $cost(a) \leftarrow$ ROUND$\big(d(a, S) * z\big)$
16:                    $q$.PUSH$\big(\pi + a, cost(\pi) + cost(a)\big)$
17:        if $q$.EMPTY() then return failure
```

As a preprocessing step, we compute text embeddings for all the grounded actions in the domain and store them in a hash table. This table is also used for storing the suggestion embeddings during search to avoid making redundant calls to the LLM. If a suggestion has already been encountered we simply use its stored embedding, otherwise we call the LLM for its embedding and store it in the table for future reference.

Beginning with an empty plan that costs 0 (line 6): If the plan achieves the story goal and is explained, it is returned as a solution (line 10). Here, p.goal is a disjunction of the author's goals—conditions in the author utility expression that evaluate to a higher utility than the initial state. If the plan is not a solution, the system prompts the LLM for suggestions about what should happen next (line 12) using the prompt text described earlier. Each action that is possible in the current state (line 14) is then assigned a cost according to its distance to the suggestions (line 15). Each new plan ($\pi + a$) is added onto the search queue, with the cost of the new action being added to the cost of the plan (line 16). Search continues with the next lowest-cost plan in the queue, until either a solution is returned (success), or the queue is emptied (failure).

E. Termination

Normally, Sabre evaluates whether a node is explained (line 9) by checking if an explanation for that node has been found in the current search. However, since this search is only considering actions that are possible (line 14), it would not necessarily find all the explanations needed to identify solutions. To ensure that solutions are identified correctly here, the check on line 9 instead uses a precomputed lookup table of all explained nodes, which were logged during a previous search.

IV. Evaluation

We argue that Sabre wastes effort when it visits nodes that are not explainable (or, actions that don't make sense for the characters who take them, given the context of the story). We consider it good when the planner visits any node that ultimately becomes explained, and bad when it visits nodes that do not. It is impossible to completely predict whether or not a node will ever become explained, because the planner can search indefinitely and may at some future point discover new explanations for earlier actions. We must use an estimate to measure explainability; for this study we consider an action "explained" if it has become explained after a complete search of one depth higher than the depth of the first solution. For each of the benchmark problems listed in Table I we ran a complete search of one depth higher than the solution depth, recording all explained nodes. These logs were used both to terminate the uniform cost searches, as discussed above, and to count the explained nodes visited by all the searches.

We test whether GPT's suggestions caused the planner to visit a higher ratio of explained to unexplained nodes compared to a baseline cost function which visits them arbitrarily. The baseline (UCS_1) is a uniform-cost search with a cost function of 1, i.e. breadth-first search. UCS_1 breaks ties deterministically based on the order in which the actions are provided in the domain, so we use an average over 10 runs, randomly shuffling the action order each time. Importantly, we also limited UCS_1 to only the *possible* nodes, so UCS_1 and UCS_GPT explore the exact same problem space. The "explainedness" of the baseline search estimates the overall ratio of explained to unexplained nodes that UCS_GPT could have visited. We can also measure progression toward the story goal by comparing the number of nodes visited by each method. Certainly breadth-first search is not a difficult baseline to beat, but we find these results valuable because they show the GPT cost function doing both of the things we need it to do simultaneously.

Sabre unfortunately cannot visit *only* the possible nodes; it is burdened with verifying explanations on its own, so it must also visit "epistemic" nodes, representing actions characters believe to be possible. In the results presented below, UCS_1 and UCS_GPT visit significantly fewer nodes than Sabre because Sabre has a larger space to explore; we are not suggesting these algorithms could replace Sabre. We include these numbers in the table because they are informative; they demonstrate how much extra work Sabre must do to account for characters' wrong beliefs and impossible plans, and how the ratio of explained to unexplained nodes changes in the epistemic parts of the search space.

We searched each problem using UCS_GPT, stopping either when a solution is found or when a maximum node limit is reached (5,000 nodes), recording all nodes visited. The node limit was necessary because some of these problems are extremely large and it would have been too costly to make all the OpenAI API calls needed to finish the search. For the same reason, we did not run multiple UCS_GPT searches for each problem and average them together, as would have been ideal. We take this data only as preliminary evidence for how the search might go, acknowledging that running it again would produce somewhat different results. UCS_1 and Sabre searches were limited to 1 million nodes and averaged over 10 runs.

A. Benchmarks

We use a set of narrative planning benchmark problems collected from a variety of sources and adapted to Sabre's format [33]. Table I lists the problems along with size information. Some problems have multiple instances, such as Grandma Lose and Grandma Win, which have different criteria for the story ending. For example, in the "Win" variant of Grandma, the planner only accepts solutions where the hero achieves his goal; in the "Lose" variant it also accepts stories where the hero is thwarted instead.

Problem	Characters $|C|$	Fluents $|F|$	Actions $|A|$	Triggers $|T|$	Solution Depth
Basketball	4	93	168	192	3
Bribery	3	25	27	0	2
Deer Hunter	3	42	28	76	6
Fantasy	4	85	76	141	5
Grandma Lose	4	96	800	952	3
Grandma Win	4	96	800	952	5
Hospital	4	67	102	196	4
Jailbreak Lose	3	46	106	54	4
Jailbreak Win	3	46	106	54	6
Raiders	3	29	35	66	6
Secret Agent	2	15	44	75	8
Snakebite	4	99	352	637	8
Space Lose	2	26	29	66	2
Space Win	2	26	29	66	5
Treasure	2	9	5	0	4

TABLE I
SABRE BENCHMARK PROBLEMS

A collection of benchmark problems for the Sabre narrative planner. We use the default configurations for each problem according to the technical report [33]. The problems are available on GitHub (https://github.com/sgware/sabre-benchmarks).

Problem	Planner	Nodes Visited	Explained
Basketball	Sabre	4,050,435	8.1%
	UCS_1	1,262	16.6%
	UCS_GPT	31	90.3%
Bribery	Sabre	39	10.3%
	UCS_1	91	9.9%
	UCS_GPT	39	12.8%
Deer Hunter	Sabre	96,310	4.4%
	UCS_1	62,688	9.1%
	UCS_GPT	2,300	19.4%
Fantasy	Sabre	80,654	48.9%
	UCS_1	75,106	62.5%
	UCS_GPT	640	81.4%
Grandma Lose	Sabre	15,307	11.6%
	UCS_1	1,155	36.9%
	UCS_GPT	37	67.6%
Grandma Win	Sabre	5,963,346	1.9%
	UCS_1	92,218	36.3%
	UCS_GPT	734	68.1%
Hospital	Sabre	335,016	14.4%
	UCS_1	75,746	51.2%
	UCS_GPT	532	74.8%
Jailbreak Lose	Sabre	211,775	38.9%
	UCS_1	12,305	24.3%
	UCS_GPT	2,851	*21.1%*
Raiders	Sabre	9,691	1.6%
	UCS_1	1,513	32.7%
	UCS_GPT	141	39.0%
Secret Agent	Sabre	780	23.1%
	UCS_1	326	43.9%
	UCS_GPT	60	*25.0%*
Space Lose	Sabre	63	48.9%
	UCS_1	14	85.7%
	UCS_GPT	6	*83.3%*
Space Win	Sabre	2,913	11.5%
	UCS_1	1,280	65.1%
	UCS_GPT	445	*59.3%*
Treasure	Sabre	125	32.1%
	UCS_1	22	59.1%
	UCS_GPT	10	80.0%

TABLE II
SEARCH RESULTS FOR UCS_GPT ON BENCHMARK PROBLEMS

The number of Nodes Visited by each planner for each problem, and the percent of those visited which are Explained. Sabre includes all epistemically accessible nodes and uses an A* heuristic search. We ran the searches with three different heuristics (the HSP additive and max heuristics [34] and the Fast Forward heuristic [35]), each averaged over 10 runs. The table shows the average of the three heuristics since there was no single best for all the problems. UCS_1 is the baseline breadth-first search, and UCS_GPT is the GPT-guided uniform cost search—both limited to non-epistemic nodes.

B. Results

Only two problems were affected by the node limits. All three planners reached their limits before solving Snakebite, and UCS_GPT reached its 5,000-node limit on Jailbreak Win. We omit these problems from the analysis. Table II shows the search results for the remaining 13 problems.

We expected that Sabre wastes a lot of effort by visiting many unexplained nodes, particularly in epistemic parts of the space. We can now support this claim with data. Compared to the non-epistemic space (UCS_1), a Sabre search typically visits a lower percentage of explained nodes—the only exceptions are Bribery (which is very small), and Jailbreak. This demonstrates why we need a way to estimate the believability of actions. There is a significant amount of search effort we could save by accurately estimating believability within one of the A* heuristics. Moreover, using LLM guidance appears to be a plausible direction for doing so. For most of the problems, UCS_GPT visited a higher percentage of explained nodes than UCS_1, and in all cases it reached the goal sooner. This indicates that the GPT cost function was guiding the planner both toward the solution and through paths of explained nodes.

For some problems, UCS_GPT did not visit a higher ratio of explained nodes than the baseline. These were Jailbreak, Secret Agent, and Space (both variants). Secret Agent resembles a single-agent pathfinding problem and has very few narrative properties. One character must navigate through several rooms, find essential items, and arrive at the other character's location. The believability of these actions mostly comes down to whether or not a particular path is valid; and as we know, LLMs are not logical problem-solvers. The LLM approach to heuristics may not be especially helpful for story domains like this. On the other hand, this is the type of problem that A* should be most capable of solving on its own. It would be interesting to see how well A* handles scaling this problem up with more than two characters, since each character adds more epistemic nodes.

Space happens to be the only one of the benchmark domains that contains any actions with no consenting characters. It has two such actions, both of which are required for the solution in both problems: *The surface begins to erupt*, and later *The surface erupts*. Though these are actions in the planning sense, the term action generally implies that someone is doing something. Since we instructed GPT to suggest "actions" that could happen next, it may have prioritized character actions over occurrences like weather events. It still suggested them eventually, but it might have done so earlier if the instruction had been to suggest "actions or events" instead.

Jailbreak is the most contrived domain. It was designed to fit a highly specific set of stories with minimal branching, and contains several actions with effects that would be better modeled as separate actions. For example, one action is *A guard confiscates the cigarettes from Ernest and sends him to the gym for punishment duties,* and another is *Roy steals the cigarettes from the kitchen. This angers the bully, who threatens to kill both Roy and Ernest.* Though the translation of the action includes these details, this translation only appears in the prompt when the action is in the node's history. Otherwise they are referred to only as "confiscate" and "steal", so the LLM would have no way of knowing these extra effects would immediately apply. This may have limited the LLM's ability to predict how the ending of the story would be reached.

However, this also suggests a simpler explanation for Jailbreak, due to the fact that these actions have unusually long translations. Since actions in the node's history act as few-shot prompting, when these actions appeared in the history it sometimes caused the LLM to make lengthier suggestions than it normally would. The problem is that we limited the number of response tokens based on an assumption for how long a typical action sentence would be. When the LLM made a long suggestion first, it would sometimes run out of tokens mid-sentence on its second suggestion; the incomplete suggestion would then be difficult to match. In fact the shortest solution contains "steal" as the first action, so this issue certainly added a lot of noise and potentially derailed the search.

Jailbreak is also unique in that the Sabre search outperforms the baseline explainedness. It also has a larger branching factor than the other problems, meaning the planner usually has more available actions to choose from. It is possible that this disproportionately affected the uniform-cost searches compared to A*, but we would need a deeper investigation into this domain to say for sure.

Barring these exceptions, the LLM method appears to do what we hoped it would. Take Basketball for example. In a space of roughly 1,200 nodes, UCS_GPT bee-lined to the solution in only 31, and it clearly did so by prioritizing believable actions, since 90% of the nodes it visited were explained. By contrast, Sabre gets lost in the weeds for over 4 million nodes in this problem, indicating that it fails to prioritize believable actions and wastes significant effort exploring nodes that cannot lead to solutions. This type of problem demonstrates the motivation for our work, and although we have yet to incorporate LLM guidance into a true search heuristic, our results indicate that this would be a promising direction.

V. Discussion

The difference in performance across the various problems is likely due to how well the different types of stories are represented in the LLM's training data. An important distinction between symbolic story planners and language-based story generators is that planners treat words as arbitrary symbols and are agnostic to their meaning, but word choice has significant implications for language-based methods. The semantic concepts represented in the domain and the words used to describe them are key to accessing the right knowledge from the LLM.

In line with others who found that LLM performance deteriorates when arbitrary symbols are used instead of relevant names [30], we found that UCS_GPT performs much worse when we obfuscate the semantics of the story world in our translations. We modified Grandma Lose by switching character names and renaming places and items, to suggest multiple genres and settings. For example, the protagonist's goal, instead of *The hero wants to have the medicine at the cottage*, becomes *The bandit wants to have the dirt at the basketball court*—with all underlying logic the same as the original problem. On average, UCS_GPT solved the original problem in 28 nodes with 75% explained, but on the obfuscated problem it took 72 nodes with only 33% explained, which is even lower than the baseline of 37%. The LLM sends the search toward unhelpful nodes that are not going to become explained, because its model of the story world is different from the planner's. For our approach to be successful, words used in the domain translations should imply a consistent genre and setting, and should have clear and unambiguous meaning that is accurately modeled in the domain.

We are encouraged by these results, but they should not be taken too generally. More experiments would be needed to show whether this methodology works similarly using different types of LLMs or with different prompts or model parameters. There are also many ways the method could be improved in terms of prioritizing explained nodes to a greater degree. Simple prompt engineering could possibly yield better results. For example, the context description could be organized by character; this might help the LLM frame the characters' individual perspectives and produce more believable suggestions.

One limitation of this method is that it relies on a text-based similarity metric, which does not always agree with semantic similarity in the context of the story. For example, the sentences *The guard attacks the bandit* and *The guard attacks the hero* have similar wording, but very different implications for the story. Conversely, the sentences *The hero runs away from the merchant* and *The hero walks to the crossroads* may reference the exact same action in the story, but they are worded quite differently and may not get recognized as matching by the algorithm we use here. Improvements to the similarity measurements or the matching algorithm might improve the method's performance overall.

There are other methods we could have tried for ranking the possible actions based on their similarity. For example, it might have been helpful to set a similarity threshold and prune actions that were not sufficiently similar to any suggestion. We also did not fine-tune the LLM on the language of each domain, mainly because we were interested in how well it would perform without doing so. Fine-tuning is certainly a possibility that could improve suggestion quality and allow more accurate similarity measurements, but care should be taken to make sure the LLM is not biased to repeat any specific actions provided to it in this training.

At this stage we have applied the LLM's guidance only in the "real" world of the story—actions whose preconditions are satisfied. In a full Sabre search, actions whose preconditions are not satisfied, but some character believes that they are, must also be explored. When this happens, Sabre keeps track of which character the node "belongs to", meaning the planner is considering the action on behalf of this character. What this means in practice is that the planner uses the character's beliefs instead of the real state when evaluating things in this node—things like whether future actions will be possible, or whether other characters will be willing to participate in them. Since the node belongs to a character, it is irrelevant whether these things are actually true; it only matters whether that character believes they are.

A simple modification of the technique used here could account for these situations: Whenever the planner is considering a node from the perspective of a character, we could just reframe the context description according to that character's perspective, otherwise prompting the LLM the same way. So, rather than describing the true initial state, we would describe the initial state according to that character's beliefs; instead of the story goal, use that character's goal; and instead of including all the events that have happened so far in the history, we would use only the events that the character observed when they happened. All of this information is readily available to the planner. Given this modified context, we would expect the resulting suggestions to lead the planner toward reasonable *character plans*. This would help the planner find usable explanations in much the same way as the current method helped it find solutions.

VI. Conclusion

Symbolic story generation methods give creators fine-grained control over what types of content can be included in generated stories. Planning algorithms can reason over large story spaces and make stronger quality assurances than reactive methods which generate content based solely on the current context. However, planning is expensive; narrative planners either struggle to maintain character believability or they use costly methods that do not scale as well as reactive systems. This is especially true for recent methods that model complex character beliefs.

In this work we explored the possibility of using a large pre-trained language model to improve the performance of a state space narrative planning algorithm. We demonstrated that GPT is capable of providing helpful suggestions to Sabre, guiding its search both toward the goal and through paths of valid nodes. For the majority of the benchmark problems, the GPT-guided search fared much better than the baseline because it was able to leverage the LLM's built-in knowledge about stories, genres, settings, people, etc. to make good guesses about which nodes to explore first. There were exceptions, but we believe these are not prohibitive, and that careful domain modeling and accurate language can mitigate most of the problems we encountered. We hope these results will inform future attempts to integrate LLM guidance into more complex search strategies.

Acknowledgement

This research was funded in part by the U.S. Department of Defense.

References

[1] J. R. Meehan, "TALE-SPIN, an interactive program that writes stories," in *Proceedings of the 5th International Joint Conference on Artificial Intelligence*, 1977, pp. 91–98.

[2] M. Lebowitz, "Story-telling as planning and learning," *Poetics*, vol. 14, no. 6, pp. 483–502, 1985.

[3] M. Mateas and A. Stern, "Structuring content in the façade interactive drama architecture," in *Proceedings of the 1st AAAI international conference on Artificial Intelligence and Interactive Digital Entertainment*, 2005, pp. 93–98.

[4] J. McCoy, M. Treanor, B. Samuel, A. A. Reed, M. Mateas, and N. Wardrip-Fruin, "Social story worlds with Comme il Faut," *IEEE Transactions on Computational Intelligence and Artificial Intelligence in Games*, vol. 6, no. 2, pp. 97–112, 2014.

[5] R. Evans and E. Short, "Versu–a simulationist storytelling system," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 6, no. 2, pp. 113–130, 2013.

[6] M. Kreminski, M. Dickinson, M. Mateas, and N. Wardrip-Fruin, "Why are we like this?: exploring writing mechanics for an AI-augmented storytelling game," in *Proceedings of the international conference on the Foundation of Digital Games*, 2020.

[7] R. M. Young, "Notes on the use of plan structures in the creation of interactive plot," in *Proceedings of the AAAI Fall Symposium on Narrative Intelligence*, 1999, pp. 164–167.

[8] M. Cavazza, F. Charles, and S. J. Mead, "Character-based interactive storytelling," *IEEE Intelligent Systems special issue on AI in Interactive Entertainment*, vol. 17, no. 4, pp. 17–24, 2002.

[9] J. Porteous, M. Cavazza, and F. Charles, "Applying planning to interactive storytelling: Narrative control using state constraints," *ACM Transactions on Intelligent Systems and Technology*, vol. 1, no. 2, pp. 1–21, 2010.

[10] J. Porteous, F. Charles, and M. Cavazza, "NetworkING: using character relationships for interactive narrative generation," in *Proceedings of the international conference on Autonomous Agents and Multi-Agent Systems*, 2013, pp. 595–602.

[11] B. Kartal, J. Koenig, and S. J. Guy, "User-driven narrative variation in large story domains using monte carlo tree search," in *Proceedings of the international conference on Autonomous Agents and Multi-Agent Systems*, 2014, pp. 69–76.

[12] R. M. Young, S. G. Ware, B. A. Cassell, and J. Robertson, "Plans and planning in narrative generation: a review of plan-based approaches to the generation of story, discourse and interactivity in narratives," *Sprache und Datenverarbeitung, Special Issue on Formal and Computational Models of Narrative*, vol. 37, no. 1-2, pp. 41–64, 2013.

[13] S. G. Ware, R. M. Young, C. Stith, and P. Wright, "The Best Laid Plans," 2014. [Online]. Available: https://nil.cs.uno.edu/projects/blp/

[14] S. G. Ware, E. T. Garcia, M. Fisher, A. Shirvani, and R. Farrell, "Multi-agent narrative experience management as story graph pruning," *IEEE Transactions on Games*, vol. 15, pp. 378–387, 2022.

[15] S. G. Ware and R. M. Young, "Glaive: a state-space narrative planner supporting intentionality and conflict," in *Proceedings of the 10th AAAI international conference on Artificial Intelligence and Interactive Digital Entertainment*, 2014, pp. 80–86.

[16] J. Teutenberg and J. Porteous, "Incorporating global and local knowledge in intentional narrative planning," in *Proceedings of the 2015 international conference on Autonomous Agents and Multiagent Systems*, 2015, pp. 1539–1546.

[17] M. Hua and R. Raley, "Playing with unicorns: AI dungeon and citizen NLP," *Digital Humanities Quarterly*, vol. 14, no. 4, 2020.

[18] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," OpenAI Blog, 2019, accessed 25 May 2023.

[19] L. Martin, P. Ammanabrolu, X. Wang, W. Hancock, S. Singh, B. Harrison, and M. O. Riedl, "Event representations for automated story generation with deep neural nets," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018, pp. 868–875.

[20] L. Yao, N. Peng, R. Weischedel, K. Knight, D. Zhao, and R. Yan, "Plan-and-write: towards better automatic storytelling," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019, pp. 7378–7385.

[21] S. Goldfarb-Tarrant, T. Chakrabarty, R. Weischedel, and N. Peng, "Content planning for neural story generation with aristotelian rescoring," in *Proceedings of the conference on Empirical Methods in Natural Language Processing*, 2020, pp. 4319–4338.

[22] H. Rashkin, A. Celikyilmaz, Y. Choi, and J. Gao, "PlotMachines: outline-conditioned generation with dynamic plot state tracking," in *Proceedings of the conference on Empirical Methods in Natural Language Processing*, 2020, pp. 4274–4295.

[23] A. Ye, C. Cui, T. Shi, and M. O. Riedl, "Neural story planning," *arXiv preprint arXiv:2212.08718*, 2022.

[24] D. McAllester and D. Rosenblitt, "Systematic nonlinear planning," Massachusetts Institute of Technology Artificial Intelligence Laboratory, Tech. Rep., 1991.

[25] T. Trabasso and L. L. Sperry, "Causal relatedness and importance of story events," *Journal of Memory and language*, vol. 24, no. 5, pp. 595–611, 1985.

[26] J. Kelly, A. Calderwood, N. Wardrip-Fruin, and M. Mateas, "There and back again: extracting formal domains for controllable neurosymbolic story authoring," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 19, no. 1, 2023, pp. 64–74.

[27] C.-H. Lin, J. Gao, L. Tang, T. Takikawa, X. Zeng, X. Huang, K. Kreis, S. Fidler, M.-Y. Liu, and T.-Y. Lin, "Magic3d: high-resolution text-to-3D content creation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 300–309.

[28] M. Heikkilä, "The algorithm: AI-generated art raises tricky questions about ethics, copyright, and security," MIT Technology Review, 2022, accessed 25 May 2023. [Online]. Available: https://shorturl.at/iswxJ

[29] S. G. Ware and C. Siler, "Sabre: A narrative planner supporting intention and deep theory of mind," in *Proceedings of the 17th AAAI International Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2021, pp. 99–106.

[30] S. Kambhampati, K. Valmeekam, L. Guan, K. Stechly, M. Verma, S. Bhambri, L. Saldyt, and A. Murthy, "Llms can't plan, but can help planning in llm-modulo frameworks," *arXiv preprint arXiv:2402.01817*, 2024.

[31] C. Siler and S. G. Ware, "Solution density and search strategy in narrative generation," *IEEE Transactions on Games*, vol. 14, no. 4, pp. 715–724, 2022.

[32] R. E. Fikes and N. J. Nilsson, "STRIPS: a new approach to the application of theorem proving to problem solving," *Artificial Intelligence*, vol. 2, no. 3, pp. 189–208, 1972.

[33] S. G. Ware and R. Farrell, "A collection of benchmark problems for the Sabre narrative planner," https://github.com/sgware/sabre-benchmarks/blob/main/report.pdf, Narrative Intelligence Lab, University of Kentucky, Tech. Rep., November 2023.

[34] B. Bonet and H. Geffner, "Planning as heuristic search," *Artificial Intelligence*, vol. 129, no. 1, pp. 5–33, 2001.

[35] J. Hoffmann and B. Nebel, "The FF planning system: fast plan generation through heuristic search," *Journal of Artificial Intelligence Research*, vol. 14, pp. 253–302, 2001.

APPENDIX