Detecting Duplicate States is Worth It in Narrative Planning with Belief

Fairoz Nower Khan, Nabuat Zaman Nahim, Stephen G. Ware, Judy Goldsmith

University of Kentucky
Lexington, Kentucky 40506 USA
fairoz.khan@uky.edu, nabuat.nahim@uky.edu, sgware@cs.uky.edu, goldsmit@cs.uky.edu

Abstract

Narrative planning algorithms generate stories as a sequence of actions that align with an author-defined goal while ensuring characters act believably, often requiring reasoning over nested beliefs. When theory of mind is represented, states include not only the factual world but also each character's beliefs about the world and other characters' beliefs, potentially to infinite depth. In such planners, detecting duplicate states can prune redundant paths in the search space, but it is unclear whether it is too computationally expensive to justify. This paper investigates the cost and benefit of duplicate state detection using the Sabre planner, which models infinitely nested beliefs deterministically. We compare two approaches: tree search, which does not check for duplicate states, and graph search, which uses a recursive equivalence algorithm to detect and avoid duplicates. We provide a polynomial-time algorithm for detecting duplicate states and empirically show that using it significantly reduces the number of nodes generated and the total planning time across several benchmark problems. These findings suggest that duplicate detection in epistemic narrative planning is both feasible and beneficial.

1 Introduction

In narrative planning, a centralized storytelling algorithm constructs a sequence of actions to tell a coherent story with a specific end condition while ensuring characters act in believable ways (Young 1999). Unlike classical planners, which generate action sequences without considering individual agents, many narrative planners account for each character's possibly incorrect beliefs and their individual goals, all while achieving an overarching author-defined objective (Riedl and Young 2010).

Narrative planning has proven useful for creating interactive stories with strict author requirements (Young et al. 2013). However, planning is computationally expensive (Helmert 2006), so there has been considerable research on how to speed up the search for valid stories.

One of the most common approaches to planning is forward state-space search (Bonet and Geffner 2001). Search begins at the initial state. The planner considers every state that can be reached by taking some number of actions in sequence. The search runs until a goal state is found, and the sequence of actions that reaches that goal state is the solution. In classical deterministic fully-observable planning, a state is an assignment of values to a fixed set of variables. States can be easily hashed, so it is easy to detect when the search has discovered two different action sequences that lead to the same state. Many classical searches can be improved by detecting and avoiding duplicate states.

When a narrative planner reasons about theory of mind—what character x believes character y believes etc.—the definition of a state becomes more complex, and it is no longer obvious that detecting duplicate states is beneficial. A node in the state space graph represents an assignment of values to each variable along with pointers to the beliefs of each character. To detect whether two nodes really represent *the same state*, we not only need to check whether they assign the same values to each variable, but we also need to check whether character x's beliefs are the same, and whether x's beliefs about y's beliefs are the same, etc. We have to check every node that is accessible via the belief relation. This means that detecting whether a node is a duplicate state is a subgraph isomorphism problem, because we have to check whether the state represented by a node *and all accessible beliefs* already exists as a subgraph in the larger state space. Checking subgraph isomorphism is NP-complete in the general case (Cook 1971), but when character beliefs are deterministic, this check can be done more efficiently. By *deterministic* beliefs, we mean that each character commits to a specific belief about every proposition, even though their beliefs may be wrong.

In this paper, we investigate this problem using the Sabre narrative planner (Ware and Siler 2021), which allows an infinitely nested theory of mind but has deterministic beliefs. We compare two approaches. The first is the original implementation of Sabre, which reduces overhead costs by not detecting duplicate states but may visit more states during search. We call this method *tree search*, since the search space can be viewed as a tree rooted at the initial state. The second method incurs the additional overhead of detecting duplicate states but may visit fewer states during search as a result. We call this method *graph search*, since its search space can be viewed as a graph where each state appears only once.

Our results show that graph search generates fewer nodes and takes less time on a suite of benchmark narrative planning problems. Based on these results, we conclude that detecting duplicate states is worth it despite the added overhead.

2 Related Work

Many researchers have used planning or planning-like systems for story generation, especially during interactive experiences where a story with a required structure needs to be rewritten in response to a player's actions. Young et. al (2013) survey several narrative planning systems. For our purposes, we broadly divide story planning systems into two types: systems that use off-the-shelf planners for storytelling and systems that use bespoke planning algorithms specifically designed to reason about story properties.

Existing general-purpose planning algorithms can be used for storytelling by encoding the constraints for a valid story into the planning problem. Examples include the use of Hierarchical Task Network planning to generate plots for the television show *Friends* (Cavazza, Charles, and Mead 2002) and using the state trajectory constraints in PDDL 3 to create an interactive version of *The Merchant of Venice* (Porteous, Cavazza, and Charles 2010) which is guaranteed to reach certain landmark states. The advantage of this approach is that as planning researchers develop faster algorithms, the scope of an interactive story that can be told also increases. The disadvantage is increased author burden. The story designer must anticipate the possible stories that can occur and constrain actions and landmarks to produce only the desired stories. As the scope of the problem increases, anticipating the large number of possible stories becomes more difficult.

The other approach is to design specialized story planning algorithms that reason about important narrative phenomena. Concepts like intention (Riedl and Young 2010; Teutenberg and Porteous 2013), conflict (Ware and Young 2014), character beliefs (Teutenberg and Porteous 2015; Ware and Siler 2021; Eger and Martens 2017), and emotion (Marsella and Gratch 2009; Shirvani, Ware, and Baker 2023) have received attention from multiple researchers. The advantage of reasoning directly about these concepts is that authors can offload this burden to the algorithm and no longer need to encode these concepts as constraints into a traditional planning problem. The disadvantage is that advances in general-purpose planning are slow to make their way into these bespoke algorithms.

Our paper is relevant to the second camp, which creates specialized storytelling algorithms. Of particular interest is the phenomenon of characters appearing to have limited observability and possibly wrong beliefs. The exact mechanics of how beliefs work in Sabre do not need to be explained here, but in short characters can observe actions happening or not (Ware and Siler 2021). When a character observes an action, they update their beliefs to incorporate the effects of the action. When they do not observe an action, their beliefs remain the same, unless the action explicitly defines changes to their beliefs. The model provides an infinitely deep theory of mind, which means it can always determine the truth value of any belief proposition, no matter how many layers of x believes y believes x believes, etc. it has. A series of studies confirmed this model matches the expectations of a human audience reading a story (Shirvani, Ware, and Farrell 2017; Shirvani, Farrell, and Ware 2018).

Built-in reasoning about beliefs is a case study in the advantages and disadvantages of bespoke narrative planners. Authors are free to define the actions that can occur in a story without needing to pay much attention to beliefs. Characters in the story will react to actions they observe and will not react to actions they do not observe. They reason about theory of mind when anticipating what other characters will do. Only when an action causes specific changes to a character's beliefs not captured by the default reasoning does the author need to encode specific propositions about beliefs into the problem. For example, if a character can lie, the author will need to explicitly define how the lie modifies the beliefs of the character being lied to.

The disadvantage of this model of beliefs is that optimizations frequently used in traditional planning algorithms cannot be directly applied. States in a general-purpose planner can usually be represented as an array of variable values. Such a state can be easily hashed, which makes detecting duplicate states easy, and avoiding duplicates often speeds up planning. This optimization cannot be directly ported to Sabre, since a state now includes not only the values of all variables but also x's beliefs, x's beliefs about y's beliefs, and so on infinitely. This paper explores how duplicate states can be detected in such a model and whether it is worth it to do so.

Other story planners have dealt with this problem differently. For example, Sanghrajka, Young, and Thorne's HeadSpace (2022) uses a 1-layer model of belief, meaning it can reason about what is actually true and what x believes, but not what x believes y believes. Teutenberg and Porteous's IMPRACTical planner (2015) does something similar but also provides a default state that represents the assumptions characters make when reasoning about belief positions deeper than 1 layer. When the depth of belief reasoning has a finite limit, it is again possible to simply represent the state as an array of values or to compile belief away entirely (Christensen, Nelson, and Cardona-Rivera 2020).

While Sabre allows an infinite theory of mind, it does not allow uncertainty. In every state, characters have exactly one state they believe the world to be in. Those beliefs may be wrong, but they cannot be uncertain. For example, x can believe y is at the store when y is actually at the office, but x cannot believe y is at the store or at the office. Some reasoning systems that have been used for games, like Eger and Martens's Ostari (2017), can represent epistemic uncertainty, and while this is a richer model, it comes at the cost of limiting the size of problems it can solve. Duplicate state detection in a system like Ostari is even more complex, and it is not clear to us whether it would be worth the cost.

3 Problem Definition

Sabre's model of narrative planning is a state-space search problem in which the goal is to identify a sequence of

actions (narrative events) that transition an initial state to a desired goal state. Each state represents the narrative world at a specific point, encoded with both factual and epistemic information (the beliefs of agents).

In narrative planning, the constructed sequence of actions must satisfy an overarching author-defined objective while ensuring that the characters act believably and in accordance with their individual goals and potentially incorrect beliefs. Formally, we define a narrative planning problem as $P = \langle C, F, A, U, s_0 \rangle$, where:

- C is a set of characters. Each character can have its own beliefs and intentions.
- F is a set of fluents, which are variables that describe the state of the world. Each fluent has a domain of possible values it can be assigned.
- A is a set of actions, similar to ADL actions (Pednault 1994). Each action has a precondition which must be true before it can occur, an effect which modifies the world state, and a set of consenting characters who must have a reason to take the action. When actions occur, they modify not only the actual world state but also the beliefs of characters. The specific rules for how beliefs are updated are described by Ware and Siler (2021), but in short, when a character observes an action they update their beliefs based on the effects, and when they do not observe an action their beliefs do not change.
- U is a set of utility functions for the characters and author. Utility is a numeric value that is a function of a state. Characters seek to maximize their personal utility values, and the planner itself seeks to maximize the author's utility value.
- s_0 is an initial state that specifies the starting values of all the fluents F and any wrong beliefs that character initially hold.

Sabre's State Graph

Sabre's search space is a graph whose nodes represent states. The graph has two kinds of edges: temporal edges that represent taking actions and epistemic edges that represent character beliefs.

Each state node stores a value for each fluent $f \in F$ and a belief for each character $c \in C$. We use the notation $s(f) = v$ to mean that in state node s fluent f has value v.

A *temporal edge* $s_1 \xrightarrow{a} s_2$ extends from state node s_1 to state node s_2 for action $a \in A$ when the precondition of action a holds in state s_1 and taking that action in that state would result in state s_2. A plan (or narrative) is a path of temporal edges through the graph. A solution is a plan that starts at s_0 and ends in any state where the author's utility is higher and where every action is explained. The full details of when actions are explained (Ware and Siler 2021) are not necessary for this paper, but in short an action is explained if it is the first step in a plan that a consenting character believes will improve their utility.

An *epistemic edge* $s_1 \xrightarrow{c} s_2$ extends from state node s_1 to state node s_2 for character $c \in C$ when the world is in state s_1 but character c believes the world is in state s_2. We use the notation $\beta(c, s_1)$ to mean the state character c believes the world is in when it is in state s_1. So when the graph contains the edge $s_1 \xrightarrow{c} s_2$, we say $\beta(c, s_1) = s_2$. Each node has exactly one epistemic edge for each character $c \in C$, even if that edge is a loop. Exactly one epistemic edge per characters means that, while characters can have wrong beliefs, they cannot be uncertain about their beliefs.

This graph allows for an infinitely nested theory of mind. If the world is in state s and we want to know whether character c_1 believes that character c_2 believes that some proposition p holds, we would evaluate p in the state $\beta(c_2, \beta(c_1, s))$.

Input: States s_1 and s_2 and list P of unordered state pairs
Output: $\top$ if s_1 and s_2 are the same state; $\bot$ otherwise

```
1:  procedure SAME(s1, s2, P)
2:      if ⟨s1, s2⟩ ∈ P
3:          return ⊤
4:      else if ∃ fluent f such that s1(f) ≠ s2(f)
5:          return ⊥
6:      else
7:          Add ⟨s1, s2⟩ to P
8:          for character c ∈ C
9:              if ¬SAME(β(c, s1), β(c, s2), P) return ⊥
10:         return ⊤
```

Figure 1: Algorithm for detecting whether two nodes represent the same state.

4 Detecting Duplicate States

While this graph makes it easy to represent theory of mind, it makes it difficult to detect when two states are *the same*. We want to say state nodes are the same if they would evaluate every possible proposition the same way. Because propositions might contain arbitrarily complex statements about beliefs, this means that all fluents must have the same value, but also that all beliefs must be the same, and that all beliefs about beliefs must be the same, and so on.

Algorithm 1 detects when two state nodes s_1 and s_2 are the same. It works by first assuming the nodes represent equivalent states and then recursively comparing their fluents and nested beliefs until it finds a counterexample which proves they are not equivalent. If it never finds a counterexample after checking every pair of nodes that should be equivalent, it returns $\top$.

Algorithm 1 first checks whether the nodes s_1 and s_2 assign a different value to the same fluent (line 4). If so, those nodes clearly do not represent the same state and the algorithm returns $\bot$. If all fluent values are the same, it recursively compares the beliefs of each character (line 8).

After verifying that all fluents are the same, but before checking character beliefs, the algorithm adds the unordered pair $\langle s_1, s_2 \rangle$ to P, a list of pairs of states that are assumed to be the same (line 7). We do this so that, if the algorithm again encounters the pair $\langle s_1, s_2 \rangle$ it can automatically return $\top$ without creating infinite recursion (line 2).

When we return $\top$ for a previously seen pair in P, we do so under the assumption that this pair has already been

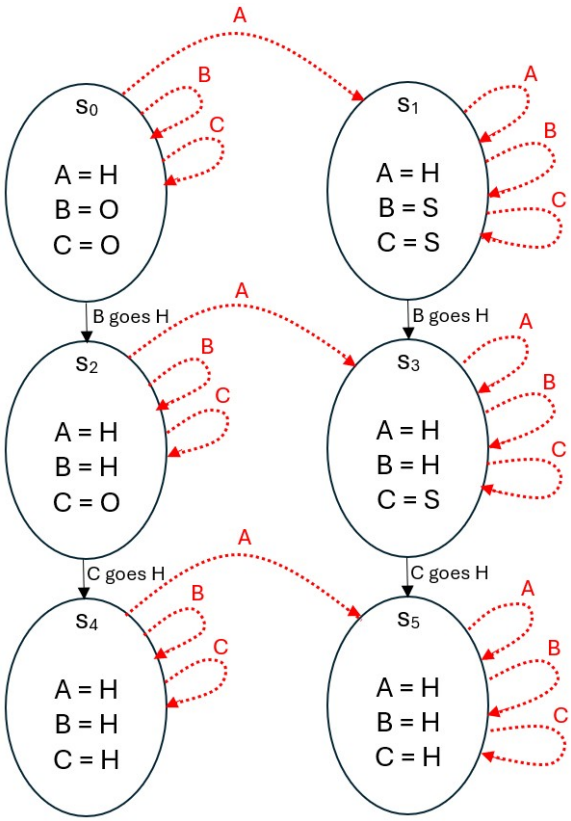

Figure 2: An example state space graph showing actions and beliefs.

verified (or is in the process of being verified) in a prior call. In other words, we want to check every pair of nodes that should be equivalent only once to avoid infinite recursion.

The set P of node pairs can be implemented as a hash table. We can add a pair of nodes to P and check whether P contains a pair of nodes in (amortized) constant time. Note that, when checking whether P contains a pair, we are checking for that exact unordered pair of nodes. The pair $\langle s_1, s_2 \rangle$ and $\langle s_2, s_1 \rangle$ are the same pair because order does not matter, but the pair $\langle s_1, s_2 \rangle$ and $\langle s_1, s_3 \rangle$ are different pairs, even if it happens that s_2 and s_3 are equivalent. In other words, hashing a node or node pair, and comparing a pair of nodes does not require checking if nodes are equivalent. In programming terms, we are checking if a pair contains exactly the same two node objects.

Two nodes are equivalent if there exists a one-to-one correspondence between their epistemic structures. Eventually, Algorithm 1 will check all pairs of states that are epistemically accessible from the original pair of states via the same sequence of characters. It will check each pair once and only once. If it finds a pair of states on the same epistemic path with different values for the same fluent, it returns $\bot$. If no such pair can be found, then the two original states can be considered the same, and it returns $\top$.

Example

Suppose Alice is at home, and Bob and Charlie are at the office, but Alice wrongly believes that Bob and Charlie are at the store. This initial state is illustrated by node s_0 in Figure 2, where $A = H$ means "Alice is at home," $B = O$ means "Bob is at the offce," etc. Epistemic edges are drawn in red dotted lines. The actual state is s_0 (Bob and Charlie are at the office), but Alice wrongly believes the state is s_1 (Bob and Charlie are at the store). This graph can be used to evaluate epistemic propositions of any depth; for example, it is true that "Bob believes Charlie believes Alice believes Bob is at the store." We evaluate this by starting at node s_0, then following the edge for B (back to node s_0), then following the edge for C (back to node s_0 again), then following the edge for A (to node s_1), and then evaluating where Bob is in node s_1, which is the store.

The edge from s_0 to s_2 is what Sabre calls a temporal edge, and it represents an action occurring. After Bob goes home, the new world state will be s_2. Now everyone agrees Bob is home, but Alice still wrongly believes Charlie is at the store when he is actually at the office (and Bob and Charlie both know Alice wrongly believes this). We can see that s_0, s_1, s_2, and s_3 are all different nodes because they assign different values to their fluents.

Now suppose Charlie also comes home, changing the world state from s_2 to s_4. If we do not bother to detect duplicate states in the graph, we could simply create states s_4 and s_5 and move on with solving the problem. However, we can potentially make our search more efficient by recognizing that nodes s_4 and s_5 are equivalent using Algorithm 1.

We start by comparing the fluent values of s_4 and s_5. All fluents have the value H in both nodes, so we add the pair $\langle s_4, s_5 \rangle$ to P. Now we need to check whether Alice has the same beliefs in both nodes. We follow the epistemic edge labeled A from both nodes, and we now check where s_5 and s_5 are equivalent. Obviously they are, since they are the same node. All other recursive checks also compare the same node to itself. Since it is not possible to find a pair of nodes which should be equivalent but are not, the algorithm returns $\top$.

Since nodes s_4 and s_5 are equivalent, we can delete node s_5 from the graph and change the temporal edge from s_3 to s_5 to go from s_3 to s_4 instead.

In Sabre, each character has exactly one outgoing epistemic edge, meaning that epistemic uncertainty is not allowed. Consequently, while duplicate state detection still involves a form of subgraph isomorphism, the problem is more structured. The constraints on epistemic edges place it in a more computationally tractable subclass of subgraph isomorphism, reducing the overhead compared to the fully general case. Detecting duplicate states in systems like Ostari (Eger and Martens 2017), which allow for epistemic uncertainty, is likely more complex.

Duplicate Detection in Polynomial Time

We now prove that Algorithm 1 runs in time polynomial in the size of the state graph produced by Sabre.

Let m be the number of nodes in the graph, and let $n = |C|$ be the number of characters. Since each node has exactly one outgoing epistemic edge for each character (including self-loops), the state graph has exactly mn epistemic edges.

We define two states s_1 and s_2 to be equivalent if and only if they would evaluate every possible proposition identically. Algorithm 1 checks this equivalence by first comparing the values of fluents and then recursively comparing beliefs by following the epistemic edges.

The algorithm uses a recursive depth-first strategy with memoization. We maintain a set P of already-compared state pairs. If $\langle s_1, s_2 \rangle \in P$, the algorithm returns $\top$ immediately. If any fluent differs, it returns $\bot$. Otherwise, it recursively compares the states believed by each character, again using memoization to prevent revisiting pairs. Since every node has exactly one outgoing epistemic edge per character, the structure formed by recursively traversing beliefs from any state is a tree of bounded out-degree n and depth at most m (since the maximum number of reachable distinct nodes is m).

Theorem 1. *Given any pair of states s_1 and s_2 in Sabre's state graph, Algorithm 1 decides whether s_1 and s_2 represent the same state in time polynomial in m and n.*

Proof. Let us analyze the worst-case complexity of comparing a single pair of states $\langle s_1, s_2 \rangle$ using Algorithm 1.

First, checking whether $\langle s_1, s_2 \rangle$ is already in P (line 2) can be done in amortized constant time assuming P is a hash table. Comparing the fluent values of both states (line 4) takes $\mathcal{O}(|F|)$ time. Then, for each character $c \in C$, we recursively compare the pair $\langle \beta(c, s_1), \beta(c, s_2) \rangle$ (line 8).

Since the belief graph is deterministic and each node has exactly one epistemic edge per character, the recursive comparisons trace through trees rooted at s_1 and s_2 with branching factor n and depth at most m (no pair of states is visited more than once due to memoization). Hence, in the worst case, the number of unique state pairs compared is $\mathcal{O}(m^2)$.

Each such comparison takes time $\mathcal{O}(|F| + n)$, and each pair is visited at most once, so the total time is $\mathcal{O}(m^2(|F| + n))$. If $|F|$ and n are bounded by m, which they typically are, the overall runtime becomes $\mathcal{O}(m^2 n)$. □

Thus, duplicate state detection via Algorithm 1 is tractable for Sabre's epistemic model and does not require full general-purpose subgraph isomorphism checking. In the theoretical worst case every pair of states will need to be considered, though in practice the number of pairs checked is typically much smaller. Having established that checking for duplicates is theoretically tractable, we now consider whether is practically useful by measuring whether and how much it speeds up planning on a suite of benchmark problems.

5 Search Methods

In this paper, we consider two methods for exploring the state-space graph: tree search and graph search. Both begin with the initial state s_0 and attempt to find a valid sequence of actions leading to a goal state, but they differ in how they manage state equivalence.

Tree Search In this approach, every new action creates new nodes in the state space graph, regardless of whether identical states might already exist. This avoids the computational cost of checking state equivalence but can result in a larger search space with many redundant nodes. Sabre version 0.7 (the current release as of this writing) uses tree search.

When using tree search, nodes are generated as needed, and a limit is placed on epistemic depth. In other words, when the state node s_1 is created and the beliefs of character c would be some other state node s_2, the edge $s_1 \xrightarrow{c} s_2$ and the node s_2 are not actually created until they are needed (i.e. until some proposition about c's beliefs is evaluated in state s_1).

Sabre searches typically specify limits on the maximum length of a plan, and a limit can also be placed on the maximum epistemic depth. For example, if epistemic depth is limited to 2, this means that when the planner tries to explain an action, it may reason about what is actually true (depth 0), what each character believes (depth 1), and what each character believes every other character believes (depth 2). An epistemic limit helps to reduce the number of nodes created during a search, but it also limits the complexity of character behavior. For example, one character might manipulate another by lying, but this requires the liar to anticipate what the other character will do when lied to, which requires a minimum epistemic depth of 2.

Graph Search This approach explicitly checks for state equivalence to detect and eliminate duplicate states. After an action creates new nodes the planner checks whether identical nodes already exist. If so, the planner replaces the new nodes with those identical nodes. This reduces the overall size of the state space graph but incurs the overhead of finding duplicate states.

When using graph search, imposing an epistemic limit will not affect the number of nodes created. Checking whether two nodes are equivalent (via Algorithm 1) requires checking all possible node pairs until every relevant pair has been checked, and this process could go to any epistemic depth. So it is theoretically possible that, even though graph search detects duplicate nodes, it could generate more nodes than tree search, since it may have to go deeper than the tree search's epistemic limit when detecting duplicates.

6 Methods

We compare the Tree Search and Graph Search methods for narrative planning with infinitely nested theory of mind but no uncertainty. We want to compare these methods on two criteria: how many nodes are generated in their respective graphs and how long it takes to expand those graphs.

We tested both methods in Sabre on a suite of 14 narrative planning benchmark problems compiled from several authors (Ware and Farrell 2023). Each problem defines a set of characters, actions, and utilities, with varying levels of complexity in terms of nested beliefs. It specifies an initial

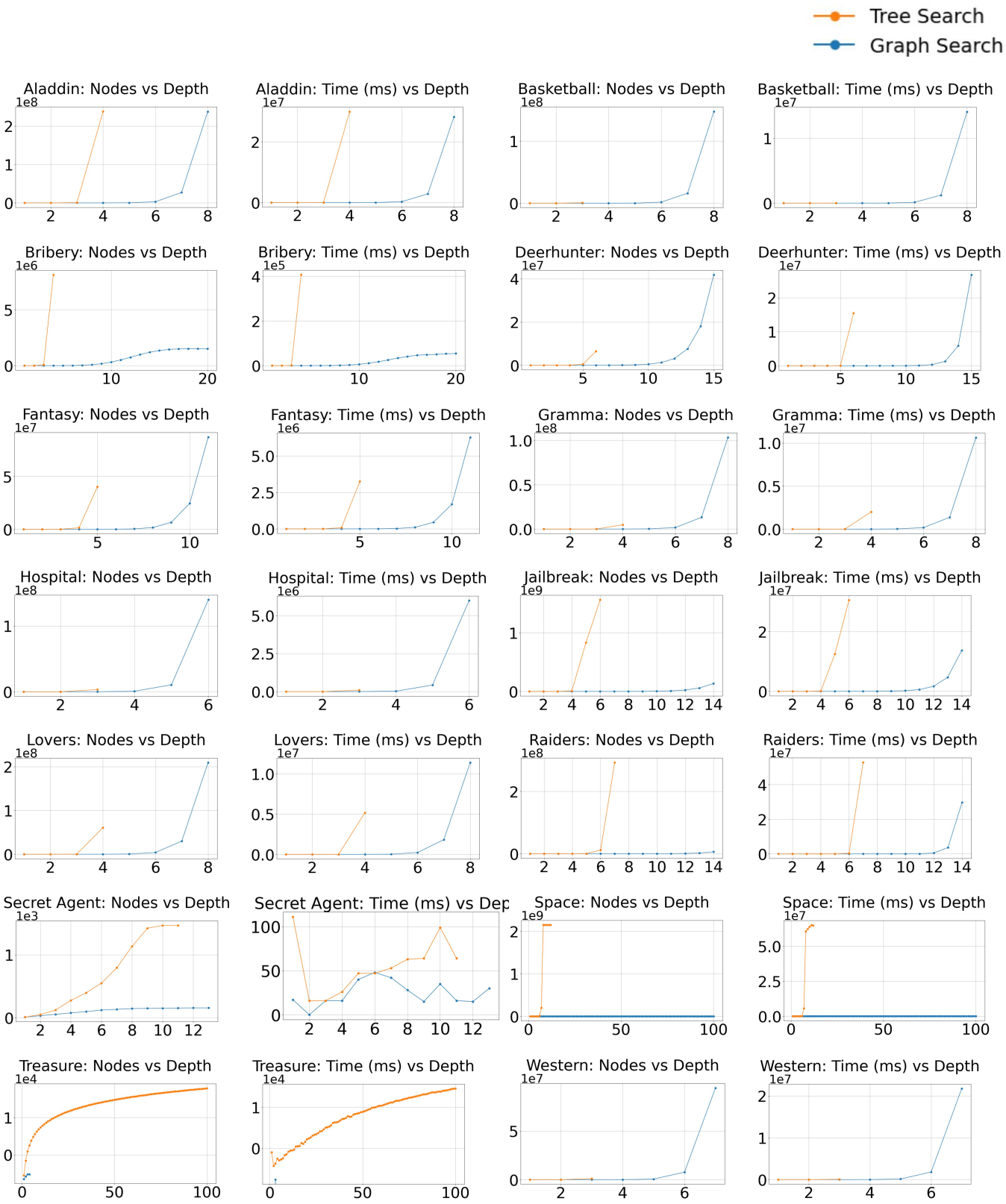

Figure 3: Comparison of Tree Search and Graph Search across 14 narrative planning benchmark problems. For each problem, two subplots are shown side by side: the left graph plots the number of nodes generated against depth (Nodes vs. Depth), and the right graph plots execution time in milliseconds against depth (Time (ms) vs. Depth). In each subplot, the x-axis represents the search depth limit, while the y-axis represents either the number of nodes generated or the time taken.

state s_0, a set of possible story actions, and a utility function for each story and character. The detailed report cited above provides background and historical context for each domain, including their origins in prior narrative planning research. This report serves as a useful reference for understanding the design and motivation behind these domains.

Experiment Design For each of the benchmark problems, we performed a breadth-first search starting at the problem's initial state. After the search, we measured the total number of nodes in the state graph and the time taken to create the graph. Below, we describe the limits placed on these searches.

There are three relevant limits that can be placed on a Sabre search. The *author temporal limit* is a limit on the number of actions in a story. The *character temporal limit* is a limit on the length of an explanation—a plan imagined by a character to justify taking an action. The *epistemic limit* is the maximum depth of theory of mind that can be used when building an explanation. The epistemic limit is discussed above in Section 5.

For each problem, we set the author temporal limit to the depth that a breadth-first search could fully expand in 24 hours when running on a computer with 4.1 GHz Intel Xeon processor and 500 GB of RAM. For the character temporal limit and epistemic limit, we used the values recommended for each problem in the benchmark suite (Ware and Farrell 2023). These values are the minimum needed to allow certain known solutions to each problem to be generated. Note that the epistemic limit only applies to tree search, as discussed above in Section 5.

The 24-hour benchmark duration and use of high-memory machines were not meant to reflect player-facing scenarios, but rather to ensure we collected as much data as we possibly could on the hardware available to us in the time we had to collect our results. We wanted to ensure results for as many benchmark problems as we could, in the hopes that these results will continue to be relevant as consumer hardware improves in future years.

7 Results

For each benchmark problem, we compared two metrics for tree search and graph search:

1. **Nodes vs. Depth** – the total number of nodes generated as a function of plan depth (x-axis: depth, y-axis: nodes generated).
2. **Time vs. Depth** – the total runtime needed to reach a given plan depth (x-axis: depth, y-axis: time in milliseconds).

The results are visualized in Figure 3. In all subplots, smaller values indicate better performance because they correspond to fewer states explored or less time spent.

Across all benchmark problems, graph search consistently generated fewer nodes than tree search for each of the depths. This result confirms our hypothesis that duplicate states are common in the Sabre search space due to the recursive and deterministic structure of character beliefs. By detecting and eliminating these duplicates, the search avoids redundant exploration of equivalent epistemic configurations and provides substantial savings.

The Time vs. Depth plots show that, despite the additional overhead of checking for duplicate states, graph search achieved shorter runtimes for each depth. The cost of duplicate detection was outweighed by the savings in node generation and search depth. For the same reasons, graph search was also able to reach greater depths within the specified time limit of 24 hours for all 14 problems.

The results demonstrate that, in the context of Sabre's structured epistemic graphs, detecting duplicate states is not only theoretically tractable but also practically beneficial. In this setting, fewer nodes directly translate into faster runtimes and deeper achievable searches, suggesting that the overhead of duplicate detection is more than offset by the pruning of redundant search paths.

While results may differ in more complex epistemic settings (e.g., those with uncertainty or multiple possible beliefs per character), within Sabre's framework, graph search clearly outperforms tree search in both efficiency and scalability.

8 Limitations

While our results suggest that detecting duplicate states improves performance in narrative planning with nested theory of mind, the following limitations should be considered when interpreting our findings. The primary limitation is that our algorithm depends on the assumption that the model is deterministic. In our model, every action has exactly one outcome and beliefs are deterministic, so characters can have wrong beliefs, but every character has exactly one belief state. This assumption excludes epistemic uncertainty and stochastic outcomes, which simplifies the structure of the belief graph and makes duplicate detection more tractable. Extending the model to handle stochastic outcomes, as in a Markov Decision Process, where actions could have multiple outcomes, or allowing characters to maintain uncertain beliefs would complicate state comparison, making the duplicate detection harder and increasing computational costs.

Furthermore, our evaluation is conducted on a specific suite of benchmark narrative planning problems. While these benchmarks are representative of common storytelling scenarios, they may not capture the full range of complexity encountered in broader narrative domains, especially those with large character sets, complex action dependencies, or deeper nesting of beliefs. It remains an open question whether the performance benefits of graph search hold consistently as problem complexity scales. These limitations point to promising directions for future work.

9 Conclusion

Our experiments show that in Sabre's deterministic, infinitely nested theory-of-mind framework, graph search with duplicate state detection consistently generates few nodes and takes less time than tree search without duplicate detection across all tested benchmarks. By avoiding redundant exploration of equivalent epistemic configurations,

graph search achieves substantial efficiency gains despite the added overhead of equivalence checking. These findings demonstrate that, at least for deterministic narrative planners like Sabre, the benefits of duplicate detection outweigh its costs, suggesting that similar techniques could improve scalability in other structured epistemic planning systems.

10 Acknowledgments

This material is based upon work supported by the U.S. National Science Foundation under Grant No. 2145153 and the U.S. Army Research Office under Grant No. W911NF-24-1-0195. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation or the Army Research Office.

References

Bonet, B.; and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence*, 129(1): 5–33.

Cavazza, M.; Charles, F.; and Mead, S. J. 2002. Character-based interactive storytelling. *IEEE Intelligent Systems special issue on AI in Interactive Entertainment*, 17(4): 17–24.

Christensen, M.; Nelson, J.; and Cardona-Rivera, R. 2020. Using domain compilation to add belief to narrative planners. In *Proceedings of the 16th AAAI conference on Artificial Intelligence and Interactive Digital Entertainment*, 38–44.

Cook, S. A. 1971. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, 151–158.

Eger, M.; and Martens, C. 2017. Practical specification of belief manipulation in games. In *Proceedings of the 13th AAAI conference on Artificial Intelligence and Interactive Digital Entertainment*, 30–36.

Helmert, M. 2006. New complexity results for classical planning benchmarks. In *Proceedings of the 16th International Conference on Automated Planning and Scheduling*, 52–62.

Marsella, S. C.; and Gratch, J. 2009. EMA: A process model of appraisal dynamics. *Cognitive Systems Research*, 10(1): 70–90.

Pednault, E. P. D. 1994. ADL and the state-transition model of action. *Journal of Logic and Computation*, 4(5): 467–512.

Porteous, J.; Cavazza, M.; and Charles, F. 2010. Applying planning to interactive storytelling: Narrative control using state constraints. *ACM Transactions on Intelligent Systems and Technology*, 1(2): 1–21.

Riedl, M. O.; and Young, R. M. 2010. Narrative planning: balancing plot and character. *Journal of Artificial Intelligence Research*, 39(1): 217–268.

Sanghrajka, R.; Young, R. M.; and Thorne, B. 2022. Headspace: incorporating action failure and character beliefs into narrative planning. In *Proceedings of the 18th AAAI conference on Artificial Intelligence and Interactive Digital Entertainment*, 171–178.

Shirvani, A.; Farrell, R.; and Ware, S. G. 2018. Combining intentionality and belief: revisiting believable character plans. In *Proceedings of the 14th AAAI international conference on Artificial Intelligence and Interactive Digital Entertainment*, 222–228.

Shirvani, A.; Ware, S. G.; and Baker, L. J. 2023. Personality and emotion in strong-story narrative planning. *IEEE Transactions on Games*, 15(4): 669–682.

Shirvani, A.; Ware, S. G.; and Farrell, R. 2017. A possible worlds model of belief for state-space narrative planning. In *Proceedings of the 13th AAAI international conference on Artificial Intelligence and Interactive Digital Entertainment*, 101–107.

Teutenberg, J.; and Porteous, J. 2013. Efficient intent-based narrative generation using multiple planning agents. In *Proceedings of the 2013 international conference on Autonomous Agents and Multiagent Systems*, 603–610.

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

Ware, S. G.; and Farrell, R. 2023. A Collection of Benchmark Problems for the Sabre Narrative Planner. Technical report, Narrative Intelligence Lab, University of Kentucky.

Ware, S. G.; and Siler, C. 2021. 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*, 99–106.

Ware, S. G.; and Young, R. M. 2014. 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*, 80–86.

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

Young, R. M.; Ware, S. G.; Cassell, B. A.; and Robertson, J. 2013. 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*, 37(1-2): 41–64.