Noname manuscript No.
(will be inserted by the editor)

Story Planning

Creativity Through Exploration, Retrieval, and Analogical Transformation

Mark O. Riedl

the date of receipt and acceptance should be inserted later

Abstract Storytelling is a pervasive part of our daily lives and culture. The task of creating stories for the purposes of entertaining, educating, and training has traditionally been the purview of humans. This sets up the conditions for a *creative authoring bottleneck* where the consumption of stories outpaces the production of stories by human professional creators. The computational automation of story generation may scale up the ability to produce and deliver novel, meaningful story artifacts. From this practical perspective, story generation systems replicate the creative abilities of humans and can thus be considered instances of *computational creativity.*

Computational systems that are purported to be creative typically utilize one of three general approaches: exploration of a space of concepts, combination of concepts, and transformation of concepts. In this article we present an approach to story generation that utilize exploration, combination, and transformation. Our approach, implemented in the Vignette Based Partial Order Causal Link story planner, is an algorithm that searches through a space of possible story solutions, guided by combinations of existing story fragments called vignettes. The vignettes are made relevant to novel story generation contexts through an automated transformation pre-process. Through these processes, we show that story generation can incorporate multiple perspectives on computational creativity. Our approach is presented at both the theoretical and technical levels.

Keywords story generation, computational creativity, case retrieval, analogical reasoning

School of Interactive Computing, College of Computing
Georgia Institute of Technology
E-mail: riedl@cc.gatech.edu

1 Introduction

Storytelling is a pervasive part of our daily lives and culture. Storytelling is particularly prominent in entertainment, where stories can be viewed as artifacts to be consumed by an audience. Story also plays a role in education and training, where stories and scenarios can be used to illustrate and guide. The relevance, prominence, and importance of narrative constructs in our culture is due in part to the relationship between sense-making and narrative reasoning (Bruner, 1990; Baumeister & Newman, 1994; Gerrig, 1993, 1994; Sternberg et al., 2000). The term *narrative intelligence* (Blair & Meyer, 1997; Mateas & Sengers, 1999) was coined to refer to human – and computational – ability to reason about, structure, and communicate through narrative. The ability to create and to tell a story is a prime example of narrative intelligence as an important and innate skill in humans.

Computational narrative intelligence promises to replicate human creative abilities to the benefit of a number of computational applications from entertainment, games, education, and training. A computational system that reasons about narrative may be able to address the creative content bottleneck by *scaling up* and *scaling in* the ability to produce and deliver narrative content for entertainment or training. Scaling up is the principle of providing goods and services to more people or doing so more often. Scaling in is the principle of customizing goods and services to the desires, needs, and interests of individuals. In the case of story generation, the ability to generate stories dynamically or on a per-session basis (once per time a user engages with the system) can create stories on-demand for novel contexts and customize narrative structure based on a user's needs, desires, and preferences.

This "practical computational creativity" perspective is essential when we consider systems that must be able to scale up and scale in. However, there is value in understanding creativity in general in order to inform the creation of new algorithms and techniques. Creativity, in the general sense, can be roughly divided into the following classes (Boden, 2004, 2009):

– *Exploratory creativity*: the process of exploring a given space of concepts.
– *Combinatorial creativity*: the process of combining existing concepts into new concepts.
– *Transformational creativity*: the process of "changing the rules" which delimit a conceptual space.

The boundaries between these classes are not necessarily always distinct (Boden, 2009).

We believe that a general solution to practical story generation requires elements from each category. There can be many algorithms for automatically generating narrative structures employing the concepts of exploration, transformation, and combination. In this article, we explore one specific, multistage approach to automatically generating narrative structures. Specifically, we look to story generation as the problem of crafting a structured sequence of events that can be told to a recipient. Focusing on practical computational creativity, we formulate the problem of generating narrative structures as a computational "search" for the right story in a space of many possible alternative stories. The search is guided by knowledge in the form of prior story fragments called *vignettes* that encode human creative intuition in the form of explicit examples of "good" narrative situations. These vignettes can be re-combined and transformed to address novel storytelling contexts.

The contribution in this article is an approach to story generation that employs exploration, combination, and transformation. The algorithms presented in this article are described at a high-level in order to address the perspective on creativity and narrative intelligence. The algorithms have been implemented in proof-of-concept form, although the absence of large libraries of narrative vignette fragments have limited real-world applicability.

The article is organized as follows. First, background material is overviewed, providing a brief primer on relevant technologies we are building from. Next, we present a framework for creativity story generation as exploration of a space of possible narrative solutions. This framework is implemented in the next two sections by an algorithm for planning narrative structures and a technique for transforming vignettes. We follow the presentation of the algorithms with general discussion of creativity and narrative intelligence issues.

2 Background

In artificial intelligence (AI), a common approach to solving problems is to *search* for the solution. A search problem requires (a) a mathematically defined space of possible solutions (of which some are actual solutions and others are not), (b) rules choosing what order to inspect elements in the space, and (c) rules for determining their candidacy as actual solutions. As enumerating the entire space is tantamount to solving the problem and is thus typically intractable, AI search algorithms typically enumerate the possible solutions just before they are inspected and evaluated. Thus an important aspect of AI is *how* to move through the space of possible solutions. Knowledge can be employed by the algorithm to answer the question: "what possible solutions should I visit next?"

We view story generation as a problem-solving activity where the problem is to create an *fabula* that meets a set of given pragmatic and aesthetic constraints. The fabula of a narrative is the chronological ordering of events that can be inferred to have happened in the story world (Bal, 1998). The fabula is what the narrative is about and what happens in the story world. Note that we are not considering discourse, which is *how* the narrative is told. To generate a fabula, a system must solve the *fabula generation problem*:

> Find a sound, coherent, and believable sequence of events that transform the world in a way that mets a set of pragmatic and aesthetic constraints.

A *sound* fabula is one in which, under the assumption that the world cannot change in unpredictable ways, no event in the fabula violates the "physics" of the story world. That is, all events are causally justified by initial conditions of the story world or by prior events. A *coherent* fabula is one in which the events are causally necessary for the occurrence of significant outcome states. For example, if a story is a tragedy in which the protagonist dies, one of the significant outcomes is the death of the protagonist. Soundness and coherence are concept derived from cognitive studies of narrative comprehension (Trabasso, Secco, & van den Broek, 1984). A *believable* fabula is one in which all characters appear to be believable characters. Believability is strongly correlated with the perception that story world characters are intentional agents with well-motivated beliefs, goals, and desires. For further discussion of believability in narratives, see Riedl and Young (Riedl & Young, 2010).

Finally, pragmatic constraints are quantifiable metrics of success such as "illustrate an historical situation," or "reinforce the recipients belief in X,"[1] or "make the recipient feel suspense." Pragmatic constraints may also include classical notions of goal, such as "propositions $p_1 ... p_n$ should be true in an intermediate or conclusion story world state." (Riedl, 2009; Porteous & Cavazza, 2009). Aesthetic constraints are harder to quantify. One aesthetic constraint identified by Aristotle (1992) may be *mimesis*, the extent to which a story mimics the real world. We assume that there are means by which humans can indicate what "good" stories look like.

A fabula generator is an algorithm that solves the fabula generation problem. Historically, fabula generators tend to employ one of two different techniques: search, and case-based reasoning. In the next two sections, we overview planning – a type of search especially well geared toward fabula generation – and case-based reasoning.

2.1 Overview of Planning

Planners are search algorithms that solve the planning problem:

> Given an initial world state, a domain theory, and a goal situation, find a sound sequence of operators that transforms the world from the initial state into a state in which the goal situation holds.

```
Action: Travel (?char, ?from, ?to)
Precondition: character(?char), place(?from), place(?to),
              alive(?char), at(?char, ?from), ?from ≠ ?to
Effect: at(?char, ?to), ¬at(?char, ?from)

Action: Mortally-Wound (?attacker, ?victim)
Precondition: character(?attacker), character(?victim),
              alive(?attacker), alive(?victim),
              stronger(?attacker, ?victim),
              battling(?attacker, ?victim), ?attacker ≠ ?victim
Effect: mortally-wounded(?victim)

Action: Die (?char)
Precondition: character(?char), mortally-wounded(?char),
              alive(?char)
Effect: ¬alive(?char)
```

Fig. 1: Portion of a planning domain library.

A *domain theory* describes the dynamics of the world – how the world works and how it can be changed. The domain theory is often a library of actions where applicability criteria and world state update rules are specified through logical statements. Specifically, the *precondition* of an action is comprised of logical statements that must be established in the world ahead of time for the action to be executable and the *effect* of an action is comprised of logical statements that describe how the world will be different if the action is executed. Figure 1 shows some uninstantiated actions from a domain theory used in later examples.

There are many algorithms that solve the planning problem. In this section we highlight one particular class of planners called *partial-order planners* (POP) (Penberthy & Weld, 1992; Weld, 1994). A partially-ordered plan consists of a set of actions that are ordered according to a set of temporal constraints of the form $a_i < a_j$, creating a *partial* ordering in the sense that some actions may be unordered relative to each

[1] See Green and Brock (Green & Brock, 2000) for an argument about the persuasive properties of narrative.

other. Additionally, *causal links*, denoted $a_1 \rightarrow^c a_2$, indicate that the effects of action a_1 establish a condition c in the world necessary for action a_2 to occur. Causal links act as protected intervals during which the truth of condition c in the world must be maintained.

POP planners are refinement search algorithms, meaning they inspect a plan, identify a flaw – a reason why the current plan being inspected cannot be an actual solution – and attempt to revise the plan to eliminate the flaw. The process iteratively repairs one flaw at a time until no flaws remain. It is often the case that there is more than one way to repair a flaw, in which case the planner picks the most promising repair, but remembers the other possibilities. Should it make a mistake, the planner can *backtrack* to revisit any previous decision point. The particular way in which flaws are identified and revised in POP is analogous to cognitive planning behavior in adult humans when faced with unfamiliar situations (Rattermann, Spector, Grafman, Levin, & Harward, 2001).

The refinement search process starts with an empty plan, identifies a flaw, and produces zero or more new plans in which the flaw is repaired (and often introducing new flaws). These plans become part of the fringe of the search space, and the process is repeated by picking the most promising plan on the fringe and iterating. The algorithm terminates when it finds a plan that has no flaws or when the plan space has been exhausted.

In POP, there are two types of flaws: *open conditions*, and *causal threats*. The open condition flaw occurs when an action in the plan (or the goal state) has a precondition that has not been recognized as being satisfied by the effect of a preceding action (or the initial state). Applying one of the following strategies can repair the flaw:

(i) Selecting an existing action in the plan that has an effect that unifies with the precondition in question.

(ii) Selecting and instantiating an operator from the domain operator library that has an effect that unifies with the precondition in question.

A causal threat flaw occurs when the temporal constraints do not preclude the possibility that an action a_k with effect $\neg c$ can co-occur during the interval of causal link $a_i \rightarrow^c a_j$. Causal threats are repaired by adding additional temporal constraints that force a_k to occur before a_i (called *promotion*) or after a_j (called *demotion*). More details are provided by Weld (1994).

Planners have been employed in fabula generation largely because the partial-order plan representation used by POP planners encapsulate many of the features that appear in cognitive models of narrative (Young, 1999). The plan representation provides a formal framework to explicitly represent causal relationships between actions and reason about them on first principles. In particular, Graesser, Lang, and Roberts (1991) and Trabasso and van den Broek (1985) highlight the importance of causality in stories. Young and Saver (2001) provide neurological evidence, noting that dorsolateral prefrontal injuries simultaneously impair behavioral planning and the ability to produce "narrative account of their experience, wishes and actions" while many other cognitive abilities remain intact. This coincidence seems to hint on the functional similarity of planning and narrative generation in the human brain.

An in-depth review of story generation systems is beyond the scope of this article. However, we briefly overview relevant planning-based story generation systems. Tale-Spin (Meehan, 1976) uses a library of plan fragments to simulate goal-driven behavior for story world characters. Dehn (1981) argues that a story generation system should satisfy the goals of the human user. That is, what outcome does the user want to see? The Universe system (Lebowitz, 1987) uses plan fragments represented as hierarchical decomposition rules to select character actions that achieve the human user's desired outcome. Porteous and Cavazza (2009) elaborate on the notion of achieving the user's goals through constraints on acceptable solutions. More recent work on narrative generation attempts to balance between character goals and human user goals (Riedl & Young, 2010). Further work on story planning addresses expanding the space of stories that could be searched (Riedl & Young, 2006b).

2.2 Overview of Case-Based Planning and Analogical Reasoning

Case-based reasoning is a process whereby previous knowledge is applied to new problems (Kolodner, 1993a). The traditional case-based planning cycle (cf., Aamodt & Plaza, 1994) involves:

- *Retrieval*: selection of one or more cases from a case base that appear to have relevance to the current problem.
- *Reuse*: adaptation and application of the cases to the current problem instance.
- *Revision*: the new solution is verified and, if necessary, further modifications to the solution are made.
- *Retain*: the new solution is stored in the case base for future retrieval.

Case-based planning (Spalzzi, 2001) works by reusing previous stored plans for new situations instead of planning from scratch. In particular, *transformational multi-reuse planners* attempt to solve a problem through retrieval and combination of several cases. Francis and Ram (1995) and Britanik and Marefat (2004) describe two transformational multi-reuse planners with close ties to POP.

Case-based reasoning has been found to be related to creativity (Boden, 2004; Kolodner, 1993b). For that reason, case-based reasoning has been applied to fabula generation. Minstrel (Turner, 1994) implements a model of cognitive creativity based on routines for transforming old stories into new stories in new domains. Gervás et al. (2005) use case-based reasoning to generate novel folk tales from an ontological case base of existing folk tales. Mexica (Pérez y Pérez & Sharples, 2001) uses elements of both previous story retrieval and means-ends planning.

Closely related to case-based reasoning, analogical reasoning is the discovery of deep structural similarities between concepts. There are two types of analogy problems:

- *Within-domain*: the analogy occurs between concepts that share the same – or very similar – knowledge and can be performed using surface similarities.

– *Between-domain*: the analogy occurs between concepts that embrace dissimilar knowledge and require deep structural similarities.

Between-domain analogical reasoning often involves representing the semantics of conceptual knowledge as graphs where nodes represent symbols and arcs represent relationships between symbols. When knowledge is represented graphically, discovery of analogical similarities between two semantic concepts A and B involves a form of graph subsumption. That is, some sub-graph of A has significant similarity to some sub-graph of B.

Analogical reasoning has been applied to stories; the "Karla the Hawk" and "Zerdia" stories are often used to illustrate an analogical reasoning system's ability to find structural similarities in first-order logical representations of narratives (cf., Falkenhainer, Forbus, & Gentner, 1989; Larkey & Love, 2003).

We highlight one particular analogical reasoning system that we utilize in our work. The Connectionist Analogy Builder (CAB) (Larkey & Love, 2003) is a cognitively inspired computational model of analogy that finds correspondences between subcomponents of two graphical representations. Given two directed graph representations, CAB detects correspondences among sub-structures, producing a mapping between corresponding elements through a constraint satisfaction approach to comparison. Each node "votes" on its best correspondence with nodes in the other graph. The similarity of paths (length of path, whether the path moves with or against the direction of arcs, et cetera) strengthens or weakens the evidence for each correspondence until all nodes converge on a consensus about their best correspondence. The more structure available in the graphs, the greater the likelihood that there will be more distinguishing features. Thus analogical reasoning works best in rich knowledge domains where concepts can be represented as graphs with numerous nodes. Further details on CAB processing is beyond the scope of this paper.

3 Planning Stories as Creative Process

In this section, we lay out a theoretical framework for refinement search as a creative system in the context of fabula generation. Boden (2004, 2009) and others (Johnson-Laird, 2002; Wiggins, 2003, 2006; Riedl & Young, 2006a) note the similarities between AI search and exploratory creativity. Wiggins (Wiggins, 2003, 2006) formalizes Boden's concept of exploratory creativity in terms of search including:

– $\mathcal{U}$: a universe of possible concepts, both partial and complete.
– $\mathcal{R}$: a set of rules that defines what it is to be an artifact of the kind one is interested in creating.
– $\mathcal{T}$: a set of rules that defines how to traverse the subspace defined by $\mathcal{R}$.
– $\mathcal{E}$: a set of rules by which *value* is attributed to an artifact.

Applying Wiggins' framework to story generation, $\mathcal{U}$ is the universe of all possible narratives (Riedl & Young, 2006a). For simplicity, we only consider the fabula of a narrative, ignoring issues of discourse. Since a narrative is a sequence of events, our universe of narratives includes partial and complete narratives. We define a *complete* narrative as one that satisfactorily achieves a set of given pragmatic and aesthetic constraints. In our interpretation of Wiggins' formalization, $\mathcal{R}$ defines what narratives can be represented, $\mathcal{T}$ is the story generation algorithm, including knowledge that is utilized by the algorithm, and heuristic guidance functions, and $\mathcal{E}$ is the definition of acceptable solutions to the user. $\mathcal{E}$ is however typically unknown or unrepresentable; the pragmatic and aesthetic constraints act as an approximation of $\mathcal{E}$.

We favor a general approach where we model the story generation process as refinement search planning because (a) as discussed earlier, plans are reasonable representations for narratives, and (b) refinement search algorithms can be thought of as "walking the space" of possible narratives in search of a solution that meets certain qualities. Note that although we base our narrative generation algorithm on refinement search planning, we do not intend for the resulting plan to be executed. Rather, the plan structure is a *description* of events to eventually be told to a recipient through some means of discourse.

What does it mean for an algorithm to "walk the space" of possible narratives? Consider the space of all possible narratives. Each point in this space is a fabula, represented as a partially ordered sequence of character actions. A domain theory – a description of how the microworld works, such as a library of action templates – defines a subspace of narratives that can be *represented.* The specific algorithm that searches this space determines which of these can be visited. Figure 2 shows the conceptual configuration of these sub-spaces.

Starting with the *empty narrative* – the narrative with no events – an algorithm walks the space of all possible narratives by applying the operator, `add-event(`e`, `c`)`. This operator moves the algorithm from the currently visited narrative to an adjacent narrative in space that differs in by exactly one additional event, e. The parameter c is a set of temporal constraints that unambiguously positions e temporally relative to other events in the narrative. That is, as a refinement to the original, the event does not necessarily have to be added to the end of an event sequence. Because events can be partially ordered with respect to each other, a second operator `constrain-ordering(`e_1`, `e_2`, `c`)` is also necessary. This operator strictly enforces the temporal ordering of events e_1 and e_2 such that e_1 must strictly occur before e_2 or vice versa depending on c.

The operators `add-event` and `constrain-ordering` are abstractions; details on how to implement them depend on the specific algorithm for fabula generation. If we recast *actions* as *events*, then the flaw repair strategies of partial-order planning map to `add-event` and `constrain-ordering`. That is, each flaw repair strategy is an explicit implementation of one of the two operators. Different implementations of refinement search that have different flaw repair strategies – and consequently the specific implementation of `add-event` and `constrain-ordering` and when they are invoked – have implications on the space of narratives that can be visited and the likelihood of finding narratives that are "good" as defined by the intersection between visitable narratives and the valued

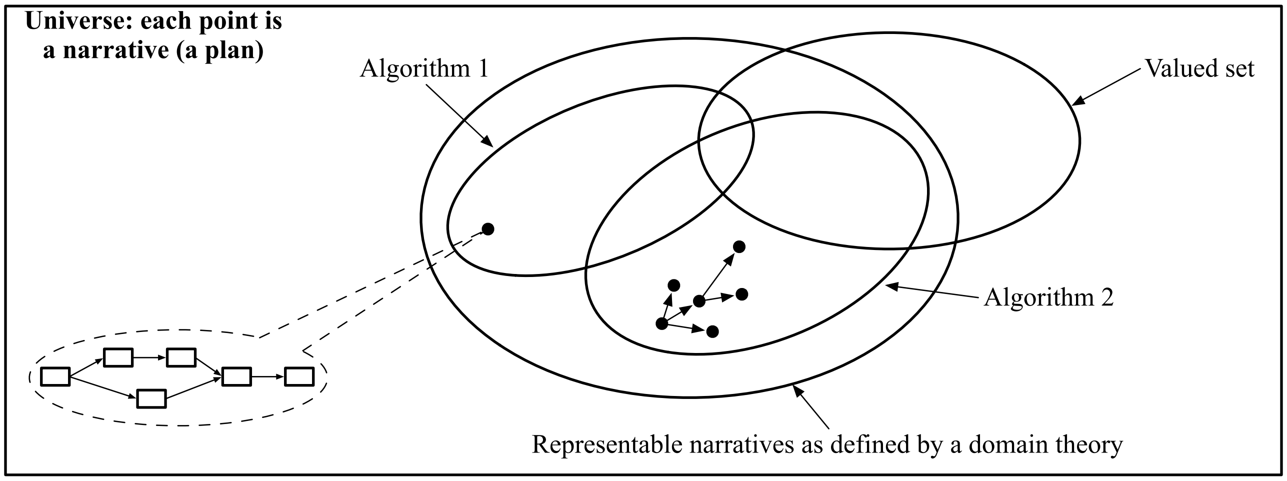
Universe: each point is a narrative (a plan)
Algorithm 1
Valued set
Algorithm 2
Representable narratives as defined by a domain theory

Fig. 2: Relationship of narrative concept space and refinement search spaces.

set. This is visualized in Figure 2 as the difference between the subspaces for algorithm 1 and algorithm 2 such that the space of narratives visitable by algorithm 2 has a greater intersection with the valued set.

Figure 2 shows an algorithm "walking the space" of narratives. As with any AI search process, we need to know the following: (a) the representation of nodes and, (b) the strategies that are used to generate successors. As discussed before, nodes are plans, complete or otherwise. Strategy (i) for repairing open condition flaws is an implementation of `add-event`. The combination of strategy (ii) for repairing open condition flaws and all repair strategies for causal threats make up the implementation of `constrain-ordering`. In Wiggins' formalization, these strategies are encoded into $\mathcal{T}$.

We argue that by changing the set of strategies that a refinement search algorithm uses to "walk the space" of creative concepts – in this case narratives – one implicitly shifts the visitable space of solutions to the benefit or detriment of the ability to find solutions that are in the valued set. In the next section, we explore an additional flaw repair strategy that guides the search of narrative space based on retrieval of previously known plot fragments called *vignettes*. There are two advantages to this approach. First, a vignette indicates choices of actions that, if incorporated into the narrative structure through refinement, are believed to result in more satisfactory results. Second, a vignette can indicate actions that should be incorporated into the narrative structure through refinement that cannot strictly be accounted for by the pursuit of soundness, meaning the new algorithm will be able to visit possible solutions not considered by a conventional POP algorithm.

4 Story Planning with Vignettes

Stories are much more than just ways of achieving an intended outcome in the most efficient manner. Stories should meet the expectations of the audience. This may mean putting in details that are aesthetically pleasing even if they are not strictly necessary. When humans write stories, they call on their lifetime of experiences as a member of culture and society, whereas a computer system that generates stories does not have access to this wealth of information. As a way of mitigating this handicap, a computer system can be provided with knowledge in the form of traces of previous problem-solving activities or a library of previous solutions, stories in this case.

We combine POP and transformational multi-reuse approaches to case-based planning and customize it to address the particulars of the fabula generation problem. Specifically, we bootstrap the planning process with a library of "vignettes" – fragments of stories that capture some particular context. In the next sections, we provide formal details on vignettes and additions to the refinement search process to effectively utilize vignettes.

4.1 Vignettes

We use the term vignette to refer to a fragment of a story that represents a "good" example of a situation and/or context that commonly occurs in stories (Riedl & León, 2008). For example, a library of vignettes would contain one or more specific instances of bank robberies, betrayals, cons, combat situations, etc. We do not presume to know how these vignettes were created, only that we have the solutions and that they have favorable mimetic qualities.

It is important to note that the library contains specific examples of these situations instead of general templates. The implication of the existence of this library is that a fabula generator does not need to "reinvent the wheel" when it comes to certain recognizable situations and thus does not need the specialized knowledge required to be able to create these specialized narrative situations. How does one know what actions should be included in the vignette and which can be left out? We use the *minimal vignette* rubric: a minimal vignette is one in which removing any one action from the vignette causes it to no longer be considered a good example of the situation and/or context it was meant to represent. That is, vignettes should not be broken up for the purposes of achieving causal sufficiency or necessity.

Computationally, vignettes are stored as partially-ordered plan fragments with some additional annotations to help the fabula generator with applicability. The core of a vignette is the set of actions that make up the vignette plus the temporal ordering that determines which actions must strictly occur before or after other actions. As vignettes are examples of salient narrative situations taken from actual examples, all actions are ground actions, meaning their parameters refer to symbols representing actual characters, places, and things. Figure 3 shows an example vignette inspired by a scene from J.R.R. Tolkien's *Silmarillion*. In the vignette, the character Fëanor attacks a Balrog, a demonic creature, wounds it twice, but is then mortally wounded and dies. The actions in the vignette are stated in the data structure, along with temporal ordering constraints (creating a partial ordering of actions) and causal relationships between actions (e.g., causal links). Additionally, a vignette has a prior state – the propositions that describe the world just before the vignette can happen.

On the right of Figure 3 is a graphical depiction of the same vignette. Boxes are actions and arrows repre-

Vignette:
Actions: 1: `Start-Battle(Fëanor, Balrog, Angband)`
2: `Wound(Fëanor, Balrog)`
3: `Wound(Fëanor, Balrog)`
4: `Mortally-Wound(Balrog, Fëanor)`
5: `Die(Fëanor)`
6: `End-Battle(Fëanor, Balrog)`
Ordering: 1 → 2, 2 → 3, 3 → 4, 4 → 5, 4 → 6
Causation: 1→`battling(Fëanor, Balrog)`→2
1→`battling(Fëanor, Balrog)`→3
1→`battling(Fëanor, Balrog)`→4
1→`battling(Fëanor, Balrog)`→6
4→`mortally-wounded(Fëanor)`→5
Prior State: `character(Fëanor)`
`character(Balrog)`
`place(Angband)`
`stronger(Balrog, Fëanor)`
`at(Fëanor, Angband)`
`at(Balrog, Angband)`

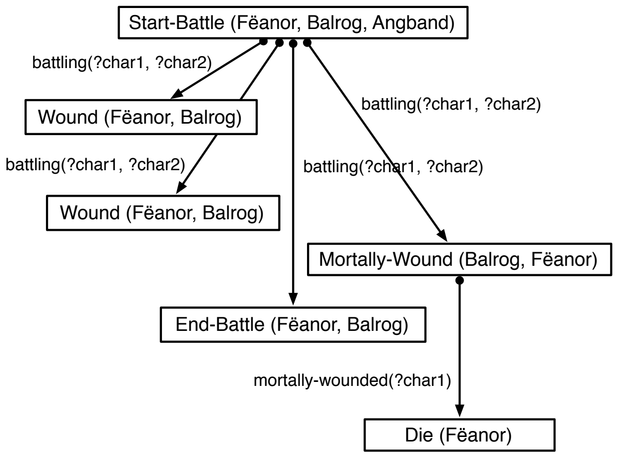

Fig. 3: An example vignette of an original and corresponding graphical representation.

sent causal links. None of the temporal constraints are shown, but readers should assume that passage of time is implicitly represented vertically.

As a plan fragment, it is possible that some actions do not have to have all of their preconditions satisfied until a fabula using this vignette is complete. This is a way of saying that it is not important how the situation is established or even why, but once the conditions are established certain things should happen with certain, partial temporal ordering.

Vignettes are examples of specific "good" narrative situations. However, we need vignettes to be flexible so that we can apply them to new story worlds with different characters, places, and things. To overcome this challenge, we pre-process vignettes to strip out specific references to characters, places, and things and replace them with variables. Figure 4 shows the same example vignette after it has been processed. Temporal and causal constraints remain the same, but we no longer have the notion of a prior state. Because our vignette now refers to variables, we do not know what state the world will be in before a vignette begins. Indeed, we do not want to know the initial state – we want the fabula generator to discover this.

Because the fabula planner is going to need to figure out how to bind variables to symbolic representations of character in the new fabula generation domain provided by the user, we need to preserve some information about what good bindings will be. Constraints determine good candidates for each variable. They are a subset of the prior state that provide ontological information about variables – propositions that do not change over time. Thus, `at(`$?char1$`, `$?place$`)` is not a constraint. Additionally, variables constraints indicate *non-co-designation* rules when variables are prohibited from referring to the same thing. Non-co-designation information is drawn directly from the domain theory.

From a planning perspective, one of the most important elements of a vignette is its effects. The effects are the union of the effects of the actions that comprise the vignette. The purpose of this information is to enable a planner to reason about what conditions can be achieved by incorporating this vignette into a narrative plan. Note that this information includes all effects of

```
Vignette:
Actions: 1: Start-Battle(?char1, ?char2, ?place)
         2: Wound(?char1, ?char2)
         3: Wound(?char1, ?char2)
         4: Mortally-Wound(?char2, ?char1)
         5: Die(?char1)
         6: End-Battle(?char1, ?char2)
Constraints: character(?char1)
             character(?char2)
             place(?place)
             stronger(?char2, ?char1)
Ordering: 1 → 2, 2 → 3, 3 → 4, 4 → 5, 4 → 6
Causation: 1→battling(?char1, ?char2)→2
           1→battling(?char1, ?char2)→3
           1→battling(?char1, ?char2)→4
           1→battling(?char1, ?char2)→6
           4→mortally-wounded(?char1)→5
Variable-constraints: ?char1 <> ?char2
Effects: battling(?char1, ?char2)
         ¬battling(?char1, ?char2)
         wounded(?char2)
         mortally-wounded(?char1)
         ¬alive(?char1)
```

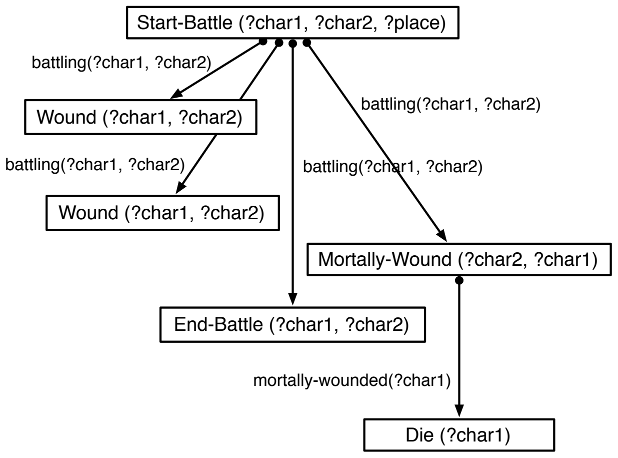

Fig. 4: The example vignette ready for use in a fabula planner.

all actions, including those that are negated by actions in the vignette that temporally occur later; the example vignette can be used to achieve both `battling` and ¬`battling`.

The advantage of instantiating a vignette in a narrative plan instead of searching for a sequence of individual actions is that this vignette captures an aesthetic relationship that cannot be represented in a plan data structure nor reasoned about by a general problem solving planner. The example vignette asserts that there is an aesthetic reason for `Wound(`$?char1$`, `$?char2$`)` to be included in the sequence twice, even though there is unlikely to be causal necessary for the action to be performed twice.

4.2 Vignette-Based Fabula Generation

The Vignette-Based Partial Order Causal Link (VB-POCL) planner is a form of transformational multi-reuse planning in the sense that it modifies the standard partial-order planning algorithm with a third strategy for repairing open condition flaws:

(iii) Retrieve and reuse a vignette that has an effect that unifies with the precondition in question.

In VB-POCL there are three strategies for refining a plan that has actions with preconditions that are not causally satisfied. The first two strategies, (i) and (ii), are described in Section 2.1. The third strategy (iii) is responsible for retrieval and reuse of vignette information.

Strategy (iii) does not strictly insert actions from the vignette into the current plan. Instead, it uses the vignette to guide future insertions and reuses of actions. That is, the vignette retrieval does not implement `add-event` nor `constraint-ordering`, but rather the vignette provides information about what future `add-event` and `constraint-ordering` operations should be chosen because those choices have been considered "good" in the past.

```
VB-POCL ($P$, $F$, $\Lambda_a$, $\Lambda_v$)

The VB-POCL algorithm takes a plan that is a partial solution to the problem (or the empty plan)
$P$, a set of flaws $F$ evidencing why $P$ cannot be a solution, a library of un-instantiated operators $\Lambda_a$ that
represent templates of actions that characters can take in the world, and a library of vignettes $\Lambda_v$.

  I. Termination: if $F = \emptyset$ and $P$ is sound, return $P$. Otherwise, if $F = \emptyset$ fail.
 II. Plan Refinement: Choose a flaw $f \in F$. Let $F' = F - f$. Switch on flaw type of $f$:
     A. Open condition: Non-deterministically choose:
          (i) Instantiate a new action $a_{new}$ that has an effect that unifies with the open condition. Extend a
              causal link.
         (ii) Reuse an existing action $a_{old}$ that has an effect that unifies with the open condition. Extend a
              causal link.
        (iii) Retrieve a vignette annotated with an effect that unifies with the open condition. Select an action
              that has an effect that unifies with the open condition to be the satisfier action. Add a fit flaw to
              the plan.
     B. Causal threat: Non-deterministically choose: promote or demote the threatening action (if no resolution
        exists, backtrack).
     C. Fit flaw: Choose an action $a_{next}$ from the retrieved vignette that hasn't been chosen before. Non-
        deterministically add an action of the same type as $a_{next}$ or reuse an existing action of the same type
        as $a_{next}$. Let $a_{fit}$ be the new or reused action. Add causal links and temporal constraints from the
        vignette relevant to $a_{fit}$. If $a_{next}$ is the satisfier action, extend a causal link from $a_{fit}$ the new or reused
        action to the action with the original open condition. For all preconditions of $a_{fit}$ that are open in the
        vignette, add a new open condition flaw to $F'$. If there are actions in the vignette that have not yet
        been chosen, $F' = F' \cup f$.
III. Recursive Invocation: Call VB-POCL($P'$, $F'$, $\Lambda_a$, $\Lambda_v$).
```

Fig. 5: The VB-POCL story planning algorithm.

When an open condition flaw is detected – an action exists that has a precondition that is not satisfied by a causal link – in the currently inspected plan, all three strategies are invoked. Focusing on strategy (iii), all vignettes that have an effect that can unify with the open condition are retrieved, and each retrieval is used to create a successor in the search space. Note that a vignette can be retrieved more than once if it has more than one action that can satisfy the original open condition flaw. The action in the vignette that can satisfy the open condition is called the *satisfier action.* For each retrieval, a duplicate plan is created, with the following difference: the new plan is identical to currently inspected plan except that it is annotated as having a *fit flaw*, indicating that the plan cannot be considered complete until it incorporates all the knowledge contained in the retrieved vignette. In the course of repairing the fit flaw, the prior open condition flaw will be solved (or the repair strategy will fail causing a backtrack)

A fit flaw, in turn, is repaired by selecting a single action in the vignette and then either (a) instantiating an action in the current plan of the same type but with entities bound to variables, or (b) reusing an existing action in the plan that matches the same type. The latter is necessary to avoid unintended action redundancies. A special case occurs when the action selected is the satisfier action, at which time a causal link is extended back to the action that was the source of the original open condition flaw. If all the actions in the vignette have not been considered in this way, the fit flaw is re-posted and the algorithm iterates. It may take several invocations of the fitting procedure to completely repair a fit flaw.

The iterative approach to vignette fitting may seem more inefficient than just adding all vignette actions to the plan at once. However, there are three advantages to iterative fitting. First, it is easier to recognize and avoid action repetition. Second, it allows for interleaving of repair of other flaws, which can lead to discovery of interesting plans. For example, fitting may lead to the creation of new open condition flaws that in turn are repaired through conventional planning (strategies (i) and (ii)) or by retrieving new vignettes (strategy (iii)). Third, problems in the fitting process can be identified sooner in case the strategy must be abandoned. These advantages are obtained because the algorithm is developed to maximize opportunistic discovery of new ways of combining narrative situations captured in vignettes. Examples of opportunistic discovery are:

– Working on two fit flaws at the same time and recognizing that the vignettes share actions, thus creating a novel narrative sequence that is reminiscent of several vignettes but not identical to any.
– Interleaving actions from two or more vignettes.
– Interleaving additional actions within the structure of a vignette to create a novel variation on the vignette.
– Dispersing a vignette throughout a larger fabula plan to create the appearance of a dramatic arc.

Figure 5 shows the VB-POCL algorithm in greater detail. Space constraints preclude a more in-depth description of the algorithm. However, the next section illustrates many of the features of the algorithm.

4.3 Example

To illustrate the VB-POCL planning algorithm, we provide an example of how the planner could use the vignette shown in Figure 4. Suppose we wanted a story set in the Middle Earth domain. The story world is initially in the state in which one character, Enemy, has in his possession a Silmaril – a precious magical stone. The outcome, provided by the human user, is that another character, Hero, gains possession of the Silmaril; this is a relatively simple pragmatic constraint. In the remainder of this section, we trace the planning process, describing only one of many possible paths through solution space that VB-POCL can follow. In actual execution, the planner employs all flaw repair strategies at every stage and uses a heuristic to judge the most promising branches of the search space to "walk."

The planner starts by non-deterministically choosing to satisfy the goal situation by having Hero take the Silmaril from Enemy. The domain theory requires that the possessor of an item not be alive for the item to be taken away. How does the planner establish the situation in which the Enemy becomes dead? There are several possibilities. One strategy is to use the vignette from Figure 4 by retrieving it and binding Enemy to $?char1$. Note that this strategy will eventually fail because it would require the Hero to be stronger than Enemy. However, the opposite is true in the domain. Because `stronger(`$?char2$`, `$?char1$`)` is a constraint of

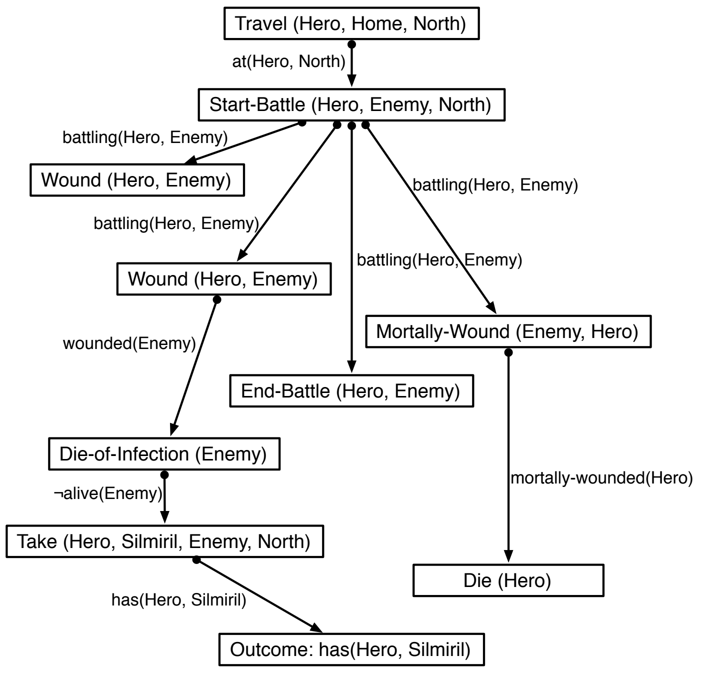

Fig. 6: A complete story plan, incorporating the example vignette from earlier.

the vignette and `stronger(Hero, Enemy)` is not true, the strategy can immediately be rejected.

Another strategy available to the planner is to choose to instantiate an action from the domain theory, `Die-of-Infection`, that causes Enemy to not be alive. Based on the domain theory, this in turn requires that Enemy be wounded. Once again, VB-POCL retrieves the vignette from Figure 4 because it has an action that can have the effect (once variables are bound) of causing Enemy to become wounded. This time the strategy will turn out to work because no constraints are violated by binding Enemy to $?char2$. Each vignette action is spliced into the new story plan one at a time, using the process of refitting described earlier.

Determining temporally *where* in the plan to splice an action is resolved by repairing causal and temporal inconsistencies (e.g. causal threats). For example, when `Die(Hero)` is spliced into the story plan, it must be temporally ordered after `Take` to avoid inconsistencies; for a character to `Take` an item, that character cannot be dead. See Figure 6.

The vignette is fairly self-contained, but the vignette action, `Start-Battle` does require that the planner establish that both Hero and Enemy are at the same place, which in this case can be a placed called North. This precondition is satisfied in the conventional way, by using conventional planning strategies to instantiate an action in which Hero travels to the North (Enemy is already there). The final story plan is shown in Figure 6.

5 Vignette Transformation

VB-POCL uses both search and combination to solve a fabula generation problem. However, until now, we have been assuming the existence of a library of vignettes

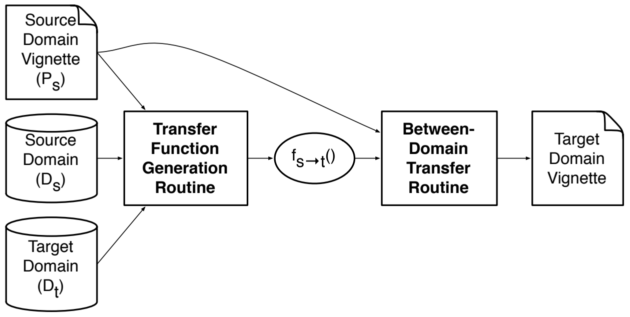
Source Domain Vignette (P_s)
Source Domain (D_s)
Target Domain (D_t)
Transfer Function Generation Routine
f_s→t()
Between-Domain Transfer Routine
Target Domain Vignette

Fig. 7: The analogical story transformation process.

that are in the story world that the novel fabula will be set in. A story world can be thought of as containing a specific set of characters, places, and things and adhere to a specific domain theory. In the previous example, the Middle Earth domain describes characters such as `Hero` and `Enemy` and operators such as `Travel` and `Wound`. As we have seen, characters and places that were not part of the original vignette can be dealt with by abstracting away specific references.

But what about different actions? For example, can we reuse a vignette set in the Middle Earth domain to generate a story set in the American Old West? Many of the actions useful for a story about the American Old West, such as holding someone at gunpoint, will not exist in the Middle Earth domain theory. We want our fabula generator to have access to vignettes from other story worlds, both similar and dissimilar.

A story world is a domain, consisting of an initial state – a set of propositions describing the world – and a domain theory – a set of all possible actions that can be performed in the world. Domains must be authored to provide a set of actions that can be performed in the world and a set of propositions that described the characters, places, objects, and relationships between them. One way to make vignettes from other domains available is to *transfer* a vignette from one domain to another by find analogies between domains and then mapping a vignette from one domain to another.

Our transformation process is visualized in Figure 7 (Riedl & León, 2009). The inputs into the process are the source vignette P_s represented as a plan fragment, the source domain D_s, and the target domain D_t. The first stage is to produce a transfer function, $f_{\mathrm{s}\mapsto\mathrm{t}}$, that maps concepts in D_s into concepts in D_t. Specifically, $f_{\mathrm{s}\mapsto\mathrm{t}}$ maps actions and ground symbols in D_s to actions and ground symbols in D_t. The second stage is to use the transfer function to map the concepts in our source narrative plan P_s into a new data structure that is a narrative plan in the target domain.

5.1 Analogical Transformation of Vignettes

We use the following *between-domain* transformation process to transfer a vignette from a source domain D_s to a target domain D_t. The target domain is the world in which a new narrative is to be generated for using VB-POCL. Consequently, all vignettes that are not already in D_t must be transferred. Vignette transformation is consequently a pre-processing stage for VB-POCL that only needs to be performed once per target domain.

5.1.1 Data Structure Preparation

In our approach, we use the Connectionist Analogy Builder (CAB) (Larkey & Love, 2003) to find analogies. CAB works with graphs, and thus we translate plan actions in to graphs. For an action, the corresponding graph has a root node with the name of the action and children nodes for the parameters of the action. We add as children to the root node a *precondition* node and an *effect* node, whose children are the propositions of the each, respectively. The graph is completed by adding selected propositions from the domain state information. The *precondition* and *effect* nodes provide structure to the graphs and also provide consistent terms that can be leveraged by CAB that help avoid very unlikely matches (e.g., it can make exact correspondences based on the keywords).

Unfortunately, many actions have very similar structures. For example, many actions take one proposition in the precondition and negate it in the effect. Consequently, the context under which an action is used in a vignette is important because it provides semantic distinctions; CAB would be unable to find the difference between any two actions with surface-level structural similarity between actions without additional contextual information. Since CAB attempts to ascertain the structural similarity, it searches for sub-graphs with distinguishing structures, context information provides a richer graphical structure that involves the *way* the action is being used in the vignetted.

Our process chooses context information to be included in the graph by incorporating world state propositions of the story world as it would exist just before the action would be executed. An example of a graph can be found in Figure 8. The white nodes at the top of the figure capture the structure of the action itself, with preconditions and effects. Gray nodes represent ground symbols. The white nodes at the bottom of the figure represent contextual state information. Finally, the small white nodes capture cardinality information, 1, 2, ..., relating predicates and parameters. For example, the ordering of parameters in `stronger(king2, princess)` is important. The contextual state information is not part of the vignette but added later.

5.1.2 Generating the Transfer Function

The transfer function is a mapping of actions in the source domain to actions in the target domain: $\{a_{s_1} \Leftrightarrow a_{t_1}\} ... \{a_{s_n} \Leftrightarrow a_{t_n}\}$. To create the mapping, our process iterates over the set of actions in the source vignette according to a total ordering of actions. If the vignette is partially ordered, it is first converted to a totally ordered sequence.

First, the algorithm obtains the current story world state for the source domain and target domain to be used as context information. On the first iteration, this is the initial state of the source and target domains, respectively. Then, for the next action in the source vignette, it invokes the `find-best` sub-procedure, which computes the action in the target domain that best corresponds to the source action, given the current states for the source and target domains. If `find-best` returns a target action, we create an entry in our mapping and then update the states for source and target domain by applying the effects of the source action and target action, respectively. This process repeats until there are no actions remaining in the source vignette.

The `find-best` routine is responsible for finding the best action in the target domain for an action in the

Action:
Head: `Capture (king2, princess)`
Precondition: `not(captured(king2)),`
`not(captured(princess)),`
`stronger(king2, princess)`
Effect: `captured(princess),`
`captured-by(princess, king2)`

Context knowledge:
`stronger(king1, king2), enemies(king1, king2),`
`daughter(princess, king1), has(king2, spear1)`

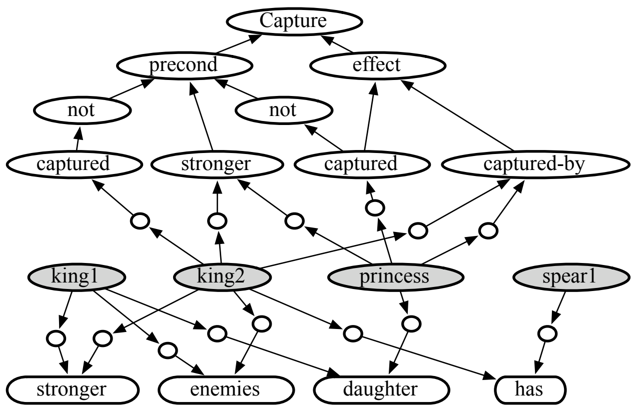

Fig. 8: An action, Capture(king2, princess) in declarative and graphical form.

source domain. It is important to note that we have not developed a test for optimality, and thus there is not an exact way for finding the best mapping. However, we refer to the "best" or "optimal" action when, intuitively, that action would be chosen by a human.

We implement `find-best` as a single-elimination competition of target domain actions. The routine compares a source action with a randomly chosen pair of target actions. Paired target domain actions are merged into the same directed graph; a union of nodes from the two actions is taken and duplicates nodes are discarded if they refer to ground symbols. See Figure 9 for an example of a pair of conflated target domain actions. The conflation of target domain actions takes advantage of the way CAB uses connectionist operations to find structural commonalities between graphs. This approach forces CAB to choose which of the two head nodes in the target domain graph makes the best correspondence to head node nodes of the graph of the single source action – the head node of the source action cannot correspond to more than one node in the target graph.

The loser target action is discarded while the winner is paired against another randomly chosen target domain action. This repeats until only one target domain action remains; the last target action remaining is the answer returned by `find-best`.

Under certain conditions, however, `find-best` will not return a mapping of head nodes. Failure occurs when structural similarity between the source graph and any portion of the target graph does not meet a minimum threshold. If there is a good analogue in the target domain, the tournament routine will find it. If there is no good analogue, meaning the source vignette references an action that has no identifiable equivalent in the target domain, `find-best` will return no solution, leaving a gap in the target vignette.

5.1.3 Applying the Transfer Function

Applying the transfer function is straight-forward. The function maps source vignette actions to target domain actions on a one-to-one basis. Once the transform algorithm is applied, we have the new vignette in the target domain.

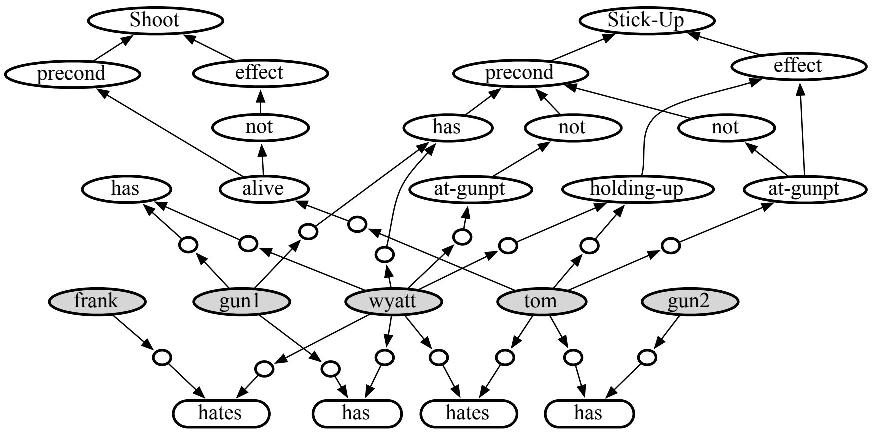

Fig. 9: Two target domain operators, Shoot(Wyatt, Tom, gun1) and Stick-Up(Wyatt, Tom, gun1), merged into a graph. For clarity, some links are left out.

In the cases where the analogical transformation stage has determined – correctly or incorrectly – that there are actions in the source vignette that have no analogical equivalents in the target domain, the transformation algorithm described above will return a target vignette with missing actions. When the vignette transformation process fails to transfer an action from the source domain to the target domain, it leaves additional conditions unsatisfied that were previously satisfied. VB-POCL, as a planner, will repair any additional unsatisfied preconditions.

5.2 Example

We demonstrate how vignette transformation transfers a vignette from a variant of the Middle Earth domain to a target domain modeling the American Old West. Whereas in Section 4.1, we could assume that specific character symbols were stripped out, our vignette transformation process requires a specific context – characters, places, and objects. The context of the source vignette is that there are two kingdoms, led by King1 and King2 who are enemies, a Princess is the daughter of King1, King1 is stronger than King2, and so on. The actions making up the source vignette are:

1. `Capture (king2, princess)`: King2 captures the princess.
2. `Marry (king2, princess)`: King2 marries the princess.
3. `Have-Child (king2, princess)`: A child is born.
4. `2-Escape (princess, child, king2)`: Princess/ child escape.
5. `Chase (king2, princess)`: King2 chases Princess.
6. `Capture (king1, king2)`: King1 captures King2.
7. `Fealty-Oath (child, king1)`: Child oath to King1.
8. `Attack (king2, child)`: King2 attacks child.
9. `Accidentally-Wound (king2, princess)`: King2 accidentally wounds Princess.
10. `Die (princess)`: Princess dies of her wounds.
11. `Kill (king1, king2)`: King1 kills King2.

The vignette is made up of a subset of the total actions available in the domain. An example of one action, `Capture`, with some accompanying initial state information is shown in Figure 8.

The target domain initial state information includes the following facts: Wyatt has gun1, Tom has gun2, Wyatt hates Tom, and Wyatt hates Frank. In the target domain, there are numerous operators. For this example, consider the following four operators:

– `Shoot (wyatt, tom, gun1)`: Wyatt shoots Tom with gun1.
– `Stick-up (wyatt, tom, gun1)`: Wyatt points gun1 at Tom.
– `Stick-up (tom, wyatt, gun2)`: Tom points gun2 at Wyatt.
– `Go (wyatt, saloon, corral)`: Wyatt goes between places.

This set of actions has been chosen in order to show a simple example, but in a real translation, we would have a larger set of actions that includes more actions and all permutations of character arguments.

The vignette translation algorithm executes as follows. The first action from the source vignette, `Capture(king2, princess)`, is selected and combined with source domain initial state information. The system randomly picks two actions from the target domain, `Go(wyatt, saloon, corral)` and `Shoot(wyatt, tom, gun1)`. CAB prefers the correspondence between `Capture` and `Shoot`. However, both `Shoot` and `Go` are very poor analogies for `Capture`. Even though CAB is forced to pick one, when none of the target actions are analogous to the source operator, the choice does not matter.

The loser of the first competition, `Go`, is removed from the list of valid target actions. Next, `Capture` is paired with the previous winner, `Shoot(wyatt, tom, gun1)`, and `Stick-Up(wyatt, tom, gun1)`. Figure 9 shows the graph of these two target domain actions. This time, CAB prefers the correspondence between `Capture` and `Stick-Up`. Note that the mapping is found despite the different number of parameters in `Capture` and `Stick-Up` and despite differences in number of preconditions and effects.

The third competition is between `Stick-Up(wyatt, tom, gun1)` and `Stick-Up(tom, wyatt, gun2)`. The only difference between target actions is the parameter order. We may wonder if it is possible for CAB to discriminate. It turns out that, because of asymmetries in context information – relations between characters and story world objects – combined with the way each operator operates on the story world state is enough to uniquely discriminate between actions.

Further trials are required before the system can finally conclude that `Capture(king2, princess)` corresponds to `Stick-Up(wyatt, tom, gun1)`. Once the tournament has settled on the best target action, the effects of the source action, `Capture(king2, princess)`, are applied to the source vignette world state and the effects of the target action, `Stick-Up(wyatt, tom, gun1)`, are applied to the target vignette world state. This prepares the respective world states for the next tournament, which is a competition to be the best correspondence to `Marry(king2, princess)` – the next successive action in the source vignette.

When the process is complete, the target vignette analogue is:

1. `Go (wyatt, saloon, corral)`: Wyatt goes between from the saloon to the corral.
2. `Stick-Up (wyatt, tom, gun1)`: Wyatt sticks up Tom.
3. `Dodge (tom, wyatt)`: Tom gets way from Wyatt.
4. `Chase (wyatt, tom, corral, bank)`: Wyatt chases Tom to the bank.
5. `Stick-Up (frank, wyatt, gun3)`: Frank sticks up Wyatt.
6. `Shoot (wyatt, tom, gun1)`: Wyatt shoots Tom.
7. `Shoot (frank, wyatt, gun3)`: Frank shoots Wyatt.

Note that the analogue vignette is shorter. In some circumstances, the target domain did not have actions that could be matched to source vignette actions. Gaps occur when best target action match to a source action does not achieve a particular correspondence threshold. However, if a original vignette is *minimal* according to the minimal vignette rubric, then there is no guarantee possible that the transfered vignette is still "good" without a human to evaluate the results of transfer.

6 Discussion

6.1 Informal Analysis of Creativity

Is VB-POCL creative? If we consider a "practical creativity" where we are only concerned about the likelihood of terminating on a narrative plan in a well-defined *valued set*, then one must be concerned with how the algorithm "walks the space" of narrative plans. Specifically, an algorithm designer must make connections between the strategies for narrative refinement and the properties of narratives in the valued set.

VB-POCL is capable of finding stories that other planning algorithms such as conventional POP planning techniques may not able to find because vignettes capture a certain degree of human authorial intuition. That is, a vignette can contain arrangements of actions that are not strictly causally necessary but are otherwise aesthetically preferable. We see this in the example vignette in Figure 4 where the double-wounding of one character precedes the death of the other character. The generated fabula shown in Figure 6 uses one causal path through the plan structure to achieve the goal situation but incorporates actions, specifically `Die(Hero)` and one of the `Wound` actions from the vignette that are not causally necessary. The conventional POP algorithm, and thus the original set of space traversal rules, does not consider any action for inclusion into the solution plan unless it is causally necessary. This is an example of a situation where changing the rules that an algorithm uses to walk the space of plans dramatically changes the narratives that can be visited. In this case, the behavior is made possible because flaw revision decisions are based on guidance from vignettes instead of strict causal necessity. As noted in (Riedl & Young, 2006a), expanding the space that can be explored provides an opportunity to find more solutions that are valuable.

Vignettes guide the local search decisions by suggesting actions that have been observed to go together in previous instances of "good" storytelling. Since this vignette potentially was transferred from an unrelated domain, one can argue that the pre-process of transferring vignettes to the new domain in which the story will be generated, *transforms* the rules on how the space of narratives can be traversed, $\mathcal{T}$. Although knowledge about how to traverse the space of narratives comes from other, unrelated domains, $\mathcal{T}$ is not dynamically changed during search. However, one could claim that some – or all – of the creativity occurred in the process of transforming vignettes, executed prior to generation.

6.2 On the Coherence of Fabula Plans

One of the interesting properties of VB-POCL is that vignette retrieval can result in story plans in which there are actions that are not causally relevant to the outcome. Trabasso and van den Broek (Trabasso & van den Broek, 1985) refer to actions that are causally irrelevant to the outcome as *dead-ends*. In the example, the causal chain involving Enemy mortally wounding Hero and then Hero dying appears to be a dead-end because those actions do not contribute to Hero acquiring the Silmaril. Dead-ends are not remembered as well as actions that are causally relevant to the outcome (Trabasso & van den Broek, 1985), suggesting that dead-ends should be avoided. A battle in which a single wound was inflicted on Enemy would have sufficed, and this is what planners such as (Weld, 1994) and (Riedl & Young, 2004) would have settled on.

However, human authors regularly include dead-end events in stories suggesting some importance to them. We hypothesize that there are certain mimetic requirements to be met in any story and that dead-ends can serve this purpose. For example, we assume that a combat scenario in which many blows of varying strengths are exchanged is more interesting than a combat in which a single blow is dealt. Interestingly, what may be a dead-end causal chain to the story planner may not be considered a dead-end by a human reader, and vice versa. That is, the reader may interpret the example story as a tragedy and consider the death of Hero as one of two primary causal chains, whereas the planners representation contains only one causal chain that leads to the human users imposed outcome (Hero has the Silmaril). More research needs to be done to create intelligent heuristics to recognize when dead-ends (from the planners perspective) are favorable, tolerable, or damaging.

6.3 Toward Practical Computational Creativity for Stories

Planning stories with vignettes is a way to increase the average length of stories that can be generated. Ideally, a planner should only have to make $O(n)$ decisions where n is the length of the plan generated. In practice planners backtrack meaning that they spend time generating action sequences that do not pan out and must return to an earlier decision point. Any effort spent on a line of reasoning that does not pan out is wasted effort. In the worst case, a planner must consider all ways of making every decision ($O(b^n)$ where b is the number of ways a decision can be made, and n is the length of the solution (Weld, 1994)). Vignettes, when selected, guide the process of adding actions to the story plan, offering up actions in chunks that are less likely to result in backtracking than if every action must be chosen independently. Of course, VB-POCL can interleave multiple vignettes during which time new issues that cause backtracking can arise; this is the price of flexibility.

Future work requires strategies for controlling the search space exploration, including heuristics for ranking solution "goodness." That is, VB-POCL currently has no understanding of how multiple vignettes add or detract from each other or the overall quality of the story being generated. Better models are needed of both narrative principles and user/readers. One way forward is to look at how narrative structure impacts cognitive and emotive state (cf., Fitzgerald, Kahlon, & Riedl, 2009) and preference modeling (cf., Yannakakis, Maragoudakis, & Hallam, 2009; Thue, Bulitko, Spetch, & Wasylishen, 2007).

VB-POCL can incorporate aesthetic considerations in two ways. First, aesthetics could be provided as constraints on the structure of the fabula. A common technique is to provide, as input into the generation problem, landmarks (Porteous & Cavazza, 2009), *author goals* (Riedl, 2009), or any other sort of constraint that effectively prunes solutions from consideration. Second, cognitive models can be incorporated directly into the iterative plan repair process in order to simulate the reader and prune, heuristically rank, or provide suggestions for improvements (O'Neill & Riedl, 2009).

6.4 Limitations of Vignette Transformation

The vignette transformation process generates new vignettes that are are analogues vignettes from other domains that are unrelated and thus unusable by the current narrative planning problem. The advantage of the approach is that source vignettes can come from heterogeneous sources with non-overlapping event vocabularities.

Our process assumes the existence of libraries of source vignettes and domain descriptions. While it is undoubtably true that there is a plethora of story material available (e.g., movies, fables, short stories, etc.), they are rarely stored in computational formats that facilitate symbolic reasoning. Independent efforts are underway to create tools and techniques that simplify the creation of story corpora (Elson & McKeown, 2007) and to mine stories from the web (Swanson & Gordon, 2008). Furthermore, recent interest in *machine reading* (Etzioni, Banko, & Cafarella, 2007) has led to techniques for automatically acquiring knowledge from primary sources without significant knowledge-engineering.

One of the limitations of our technique for analogical story transformation is that it requires the source and target domains to be represented at approximately the same level of abstraction. That is, one domain cannot use extremely abstract actions while another domain uses more primitive actions because it will be difficult to find the structural similarities between actions.

An observation we have made is that the less analogical the domains, the more gaps appear in the target story. Further, the more gaps, the less likely CAB will be able to find correspondences between operators after the gaps because the source and target state information become "out of sync." We have not yet determined how to measure the degree of analogical similarity between domains. Our system has not been tested on story world domains of significant complexity, in terms of number of characters or number of distinct propositional statements. These gaps potentially violate the *minimal vignette* assumption resulting in vignettes that do not contribute to the quality of the overall story as much as expected.

7 Conclusions

Partially ordered plans have been shown to be good representations of fabula (Young, 1999; Porteous & Cavazza, 2009; Riedl & Young, 2010). Consequently, it makes sense that refinement search, one means for generating plans, can be used to generate fabula structures. In particular, refinement search, when applied to fabula generation, can be thought of as "walking the space" of possible narrative solutions. We consider how a fabula generator can employ concepts such as combination and transformation from computational creativity to incorporate valuable knowledge contained within story fragments, achieve properties such as mimesis, and achieve efficiency gains.

In implementing our framework for creative story planning, we employ knowledge in the form of vignettes, which can be mined from existing stories and made available in computational form. Since a story generator is utilized to generate novel stories, it is often the case that the story generator is operating on a story world that is significantly different from those that have existed before. Consequently, we use a vignette transformation process to ensure VB-POCL's access to vignettes by transforming them into a form readily usable before hand. As many regard *transformational* creativity to be the most important form of creativity (cf., Boden, 2004), it may be the case that the transformation pre-process is the source of greatest creativity in our system.

We believe that computational creativity is an integral aspect of practical scaling of creative content production. In this article, we only consider storytelling content as a domain for "practical creativity." However, the prevalence of narrative in entertainment, education, and training invests narrative intelligence with considerable importance.

References

Aamodt, A., & Plaza, E. (1994). Case-based reasoning: Foundational issues, methodological variations, and system approaches. *Artificial Intelligence Communications*, *7*(1), 39–59.

Aristotle (1992). *The Poetics*. Prometheus Books, Buffalo, N.Y. T. Buckley, trans., original published 350 B.C.E.

Bal, M. (1998). *Narratology: An Introduction to the Theory of Narrative*. University of Toronto Press.

Baumeister, R. F., & Newman, L. S. (1994). How stories make sense of personal experiences: Motives that shape autobiographical narratives. *Personality and Social Psychology Bulletin*, *20*(6).

Blair, D., & Meyer, T. (1997). Tools for an interactive virtual cinema. In Trappl, R., & Petta, P. (Eds.), *Creating Personalities for Synthetic Actors: Towards Autonomous Personality Agents*. Springer.

Boden, M. (2004). *The Creative Mind: Myths and Mechanisms, 2nd Edition*. Routledge.

Boden, M. (2009). Computer models of creativity. *AI Magazine*, *30*(3), 23–34.

Britanik, J., & Marefat, M. (2004). CBPOP: A domain-independent multi-case reuse planners. *Computational Intelligence*, *20*.

Bruner, J. (1990). *Acts of Meaning*. Harvard University Press, Cambridge.

Dehn, N. (1981). Story generation after Tale-Spin. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence*.

Elson, D., & McKeown, K. (2007). A platform for symbolically encoding human narratives. In *Proceedings of the AAAI Fall Symposium on Intelligent Narrative Technologies*.

Etzioni, O., Banko, M., & Cafarella, M. (2007). Machine reading. In *Proceedings of the AAAI Spring Symposium on Machine Reading*.

Falkenhainer, B., Forbus, K., & Gentner, D. (1989). The structure-mapping engine: Algorithms and examples. *Artificial Intelligence*, *41*, 1–63.

Fitzgerald, A., Kahlon, G., & Riedl, M. O. (2009). A computational model of emotional response to stories. In *Proceedings of the 2nd Joint International Conference on Interactive Digital Storytelling*.

Francis, A., & Ram, A. (1995). A domain-independent algorithm for multi-plan adaptation and merging in least-commitment planners. In *AAAI Fall Symposium on Adaptation of Knowledge for Reuse*.

Gerrig, R. (1994). Narrative thought?. *Personality and Social Psychology Bulletin*, *20*(6), 712–715.

Gerrig, R. J. (1993). *Experiencing Narrative Worlds: On the Psychological Activities of Reading*. Yale University Press, New Haven.

Gervás, P., Díaz-Agudo, B., Peinado, F., & Hervás, R. (2005). Story plot generation based on CBR. *Journal of Knowledge-Based Systems*, *18*(4–5).

Graesser, A., Lang, K. L., & Roberts, R. M. (1991). Question answering in the context of stories. *Journal of Experimental Psychology: General*, *120*(3), 254–277.

Green, M. C., & Brock, T. C. (2000). The role of transportation in the persuasiveness of public narratives. *Journal of Personality and Social Psychology*, *79*(5), 701–721.

Johnson-Laird, P. (2002). How jazz musicians improvise. *Music Perception*, *19*(3).

Kolodner, J. (1993a). *Case-Based Reasoning.* Morgan Kaufmann Publishers.

Kolodner, J. (1993b). Understanding creativity: A case-based approach. In *Proceedings of the 1st European Workshop on Case-Based Reasoning.*

Larkey, L. B., & Love, B. C. (2003). CAB: Connectionist analogy builder. *Cognitive Science*, *27*, 781–794.

Lebowitz, M. (1987). Planning stories. In *Proceedings of the 9th Annual Conference of the Cognitive Science Society.*

Mateas, M., & Sengers, P. (1999). Narrative intelligence. In Mateas, M., & Sengers, P. (Eds.), *Narrative Intelligence: Papers from the 1999 Fall Symposium (Technical Report FS-99-01).* AAAI Press, Menlo Park, CA.

Meehan, J. R. (1976). *The Metanovel: Writing Stories by Computer.* Ph.D. thesis, Yale University.

O'Neill, B., & Riedl, M. O. (2009). Supporting human creative story authoring with a synthetic audience. In *Proceedings of the 7th Creativity and Cognition Conference.*

Penberthy, J. S., & Weld, D. S. (1992). UCPOP: A sound, complete, partial-order planner for ADL. In *Proceedings of the 3rd International Conference on Knowledge Representation and Reasoning.*

Pérez y Pérez, R., & Sharples, M. (2001). MEXICA: A computer model of a cognitive account of creative writing. *Journal of Experimental and Theoretical Artificial Intelligence*, *13.*

Porteous, J., & Cavazza, M. (2009). Controlling narrative generation with planning trajectories: the role of constraints. In *Proceedings of the 2nd International Conference on Interactive Digital Storytelling.*

Rattermann, M., Spector, L., Grafman, J., Levin, H., & Harward, H. (2001). Partial and total-order planning: evidence from normal and prefrontally damaged populations. *Cognitive Science*, *25*, 941–975.

Riedl, M. O. (2009). Incorporating authorial intent into generative narrative systems. In Louchart, S., Roberts, D., & Mehta, M. (Eds.), *Intelligent Narrative Technologies II: Papers from the 2009 Spring Symposium*, Palo Alto, CA. AAAI Press.

Riedl, M. O., & León, C. (2008). Toward Vignette-Based Story Generation for Drama Management Systems. In *Proceedings of the 2nd International Conference on Intelligent Technologies for Interactive Entertainment (INTETAIN), Workshop on Integrating Technologies for Interactive Stories.*

Riedl, M. O., & León, C. (2009). Generating story analogues. In *Proceedings of the 5th Artificial Intelligence and Interactive Digital Entertainment (AIIDE).*

Riedl, M. O., & Young, R. M. (2004). An Intent-Driven Planner for Multi-Agent Story Generation. In

Proceedings of the 3rd International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS).

Riedl, M. O., & Young, R. M. (2006a). From Linear Story Generation to Branching Story Graphs. *IEEE Journal of Computer Graphics and Animation*, *26*(3), 23–31.

Riedl, M. O., & Young, R. M. (2006b). Story Planning as Exploratory Creativity: Techniques for Expanding the Narrative Search Space. *New Generation Computing*, *24*(3), 303–323.

Riedl, M. O., & Young, R. M. (2010). Narrative planning: Balancing plot and character. *Journal of Artificial Intelligence Research*, *in press.*

Spalzzi, L. (2001). A survey on case-based planning. *Artificial Intelligence Review*, *16*(1), 3–36.

Sternberg, Robert, J., Forsythe, G. B., Hedlund, J., Horvath, J. A., Wagner, R. K., Williams, W. M., Snook, S. A., & Grigorenko, E. (2000). *Practical Intelligence in Everyday Life.* Cambridge University Press.

Swanson, R., & Gordon, A. (2008). Say anything: A massively collaborative open domain story writing companion. In *First International Conference on Interactive Digital Storytelling.*

Thue, D., Bulitko, V., Spetch, M., & Wasylishen, E. (2007). Interactive storytelling: A player modelling approach. In *Proceedings of the 3rd Conference on Artificial Intelligence and Interactive Digital Entertainment.*

Trabasso, T., Secco, T., & van den Broek, P. (1984). Causal cohesion and story coherence. In Mandl, H., Stein, N., & Trabasso, T. (Eds.), *Learning and Comprehension in Text.* Lawrence Erlbaum Associates.

Trabasso, T., & van den Broek, P. (1985). Causal thinking and the representation of narrative events. *Journal of Memory and Language*, *24*, 612–630.

Turner, S. R. (1994). *The Creative Process: A Computer Model of Storytelling.* Lawrence Erlbaum Associates, Hillsdale, NJ.

Weld, D. (1994). An introduction to least commitment planning. *AI Magazine*, *15.*

Wiggins, G. (2003). Categorising creative systems. In *Proceedings of the IJCAI 2003 Workshop on Creative Systems.*

Wiggins, G. (2006). Searching for computational creativity. *New Generation Computing*, *24*, 3.

Yannakakis, G., Maragoudakis, M., & Hallam, J. (2009). Preference learning for cognitive modeling: A case study on entertainment preferences. *IEEE Systems, Man and Cybernetics; Part A: Systems and Humans*, *39*(6), 1165–1175.

Young, K., & Saver, J. L. (2001). The neurology of narrative. *SubStance: A Review of Theory and Literary Criticism*, *30*, 72–84.

Young, R. M. (1999). Notes on the use of plan structures in the creation of interactive plot. In Mateas, M., & Sengers, P. (Eds.), *Narrative Intelligence: Papers from the AAAI Fall Symposium (Technical Report FS-99-01).* AAAI Press, Menlo Park.