A Conceptual Blending Approach to the Generation of Cognitive Scripts for Interactive Narrative

Justin Permar and Brian Magerko
{jpermar, magerko}@gatech.edu
Georgia Institute of Technology
225 North Ave NW
Atlanta, GA 30332 USA

Abstract

This paper presents a computational approach to the generation of cognitive scripts employed in freeform activities such as pretend play. Pretend play activities involve a high degree of improvisational narrative construction using cognitive scripts acquired from everyday experience, cultural experiences, and previous play experiences. Our computational model of cognitive script generation, based upon conceptual integration theory, applies operations to familiar scripts to generate new blended scripts.

1 Introduction

People regularly engage in social activities that conform to cognitive scripts, which are temporally ordered sequences of actions and events that represent a narrative experience (Schank and Abelson 1977). Our everyday experiences often involve modifications to cognitive scripts, where features from multiple scripts are meshed together to help us make sense of the world or create an interesting new narrative experience. For example, children pretend play as heroes or villains based upon situations seen in superhero films. Although this narrative process is ubiquitous, formally understanding how to combine narrative scripts to elicit new, creative scripts is an unsolved problem (Magerko et al. 2009). By better understanding how to accomplish this, we can inform approaches to computational creativity that deal with narrative understanding and generation.

Our work addresses the generation of unique scripts given two input scripts from an agent's background knowledge in the narrative domain of pretend play. Playing pretend, as a creative act, frequently necessitates the generation of scripts that are blended from cultural references, past play experiences, and common cultural scripts (Moran, John-Steiner, and Sawyer 2003). In this paper, we present an algorithm that generates new "script blends", represented as scripts. This capability can be used, for example, by an AI agent to execute scripts that are not defined a priori. Similarly, an agent could use this capability to aid script recognition during pretend play activities. Although scripts consist of causal and temporal links, both of which are critical to comprehension, here we focus on an explicitly temporal representation of activities. Our method of modeling a process that produces script blends from input scripts is based on the theory of conceptual blending.

Conceptual blending (also known as "conceptual integration theory") has been proposed as a basic cognitive operation underlying our capacity for creativity as well as mundane meaning-making (Fauconnier and Turner 2003). Pretend play is a narrative domain where numerous creative feats regularly appear, including the use of real objects as pretend objects, diegetic and extra-diegetic context switches, and script blending, such as playing cars and Godzilla wrecking the city. However, script blending is certainly not confined to pretend play, but instead occurs pervasively throughout creative works. Music composition, film pastiche and narrative sub-genres are all examples of art works created from multiple sources.

Conceptual blending is a theoretical abstraction encompassing a group of related cognitive processes, including analogical reasoning, mental modeling, and similarity (Fauconnier and Turner 1998). Fauconnier and Turner's network model of conceptual blending involves four conceptual spaces: a generic space, two inputs spaces, and a blend (output) space. In addition to the overall structure of these spaces, conceptual blending also proposes standard processes that occur for the purposes of "on-line, dynamical cognitive work" (Fauconnier and Turner 1998). This network model is a generalization of many cognitive processes that produce an output derived from two inputs, such as a process of script generation using two familiar input scripts. Conceptual blending processes that are relevant to script generation include 1) the cross-space mapping of counterpart connections between input spaces and 2) selective projection from the input spaces to the blend space. A key result of blending is that the composition of elements (in the blend space) from the input spaces results in relations not present in either input space.

To date, research related to conceptual blending, which we review in section 2, has mostly focused on metaphor and analogy. Some research has successfully utilized conceptual blending for other purposes, including interactive poetry generation with Harrell's GRIOT system and the work by Li et al. on the generation of fantastical and imaginary pretend play objects (Harrell 2005; Li et al. 2012). How-

ever, no work has yet applied conceptual blending to cognitive structures that explicitly represent temporality, such as scripts. Our primary contribution is an initial algorithm that modifies two familiar scripts to produce a blended script, a capability that can inform generalized approaches to computational generation of creative narratives.

This paper is outlined as follows. In section 2, we review related work in conceptual blending and computational models of generative processes. In section 3, we present our script generation algorithm alongside a motivating example in order to ground the remainder of the paper. In section 4, we present two additional examples illustrating details of our algorithm and approach. Lastly, in section 5 we conclude with a critical discussion of our current approach, highlight limitations of our algorithm, and present future work.

2 Related Work

Conceptual Blending

Zook et al. propose a formal model of pretend object play based on a two-space model of conceptual blending (Zook, Magerko, and Riedl 2011). The model contains a conceptual space comprised of real objects in a real domain and a second conceptual space comprised of imaginary objects in a pretend domain. They propose a process in which the last step blends attributes of real and pretend objects to produce a blended object, such as a pretend sword. Both our work and the work by Zook et al. employ a process involving substitution operations during blending, but their work does not involve inputs with a temporal dimension, such as scripts, which necessitates a significantly different approach to counterpart mapping.

Veale et al. provide a computational model of conceptual blending called Sapper that uses a semantic network to model concepts and relations between concepts and a process of spreading activation to explore the conceptual domain (Veale, O'Donoghue, and Keane 2000). Their goal during counterpart mapping is to establish a sub-graph isomorphism between the two input spaces, which allows "projection" of mapped properties between objects. Veale et al. base the means for establishing the sub-graph isomorphism on a structural mapping that is derived from Gentner's structure-mapping theory of analogy and the property of systematicity, which is the idea that particular (analogical) mappings are defined by the existence of higher-order relations (Gentner 1983). Sapper takes as input a semantic network, which does not represent the temporal nature of scripts that is central to our problem and approach.

Goguen and Harrell developed the ALLOY blending engine, noting that concepts (and relations between concepts) are a representation which does not fully support describing "the structure of things" (Goguen and Harrell 2004). Thus, they focus on a variant of conceptual blending that they call structural blending or structural integration. Semiotics, an integral part of the ALLOY blending engine, includes the notion of morphisms that map between signs in a source space and signs in a target space. ALLOY has a notion of partial mapping between elements in the source and target spaces, which is a significant differentiation from later work that treats mappings as exclusively one-to-one. Our algorithm mirrors later work that only uses one-to-one mappings.

Computational Models of Generative Processes

Holyoak and Thagard built ACME to produce analogies, which at its core is a problem of determining suitable correspondences between two inputs (1989). Holyoak and Thagard describe categories of constraints that must be respected in order to derive analogies, including 1) structural, 2) semantic, and 3) pragmatic constraints. Structural constraints specify that an isomorphism between inputs is valid if, and only if, the mapping is one-to-one for any objects and relations (see (Holyoak and Thagard 1989) for formal definitions). They use a matching process that can establish mappings between relations when only the number of arguments matches, regardless of order. Our approach uses a matching process that uses both the number and order of arguments, but slightly relaxes the requirements from exact matches for all arguments to exact matches on all arguments except one.

Falkenhainer et al. developed SME as an implementation of analogical reasoning based upon Gentner's structure mapping theory (Falkenhainer, Forbus, and Gentner 1989). SME is an implementation of analogy based on structural similarity between two representations (of situations). Structural similarity is achieved by aligning elements of two (similar) representations using two primary constraints: 1) one-to-one mapping, and 2) parallel connectivity (retaining the order of arguments for a predicate). SME, as pointed out by (Holyoak and Thagard 1989) requires an exact match of predicates (part of the predicate calculus representation favored by Forbus et al.) during mapping. Closely related to SME is MAC/FAC, where the MAC stage uses a computationally inexpensive non-structural filter to reduce computational complexity (Forbus, Gentner, and Law 1995). Our approach is closely related to SME, as we use the notions of predicate matching and parallel connectivity to assert equivalence. A key difference is our use of path representations during structural alignment.

3 Script Blending Algorithm

In order to guide this section, we present an example blend between two scripts familiar to Western culture: a stagecoach ride script, shown in Figure 1, and a guided tour bus tour script, shown in Figure 2. Our script representation is a directed acyclic graph (DAG), a restriction of a general graph structure that we require in order to establish a finite number of "paths" through a script. The script is read top-down, with an initial event beginning the script. The edges between nodes are directed, indicating a temporal ordering among actions in the script. The predicates should be interpreted as actions. Parentheses surround the argument list for the predicate, henceforth referred to as the entity list. All entities are capitalized in order to visually distinguish between entities and predicates. The last noteworthy feature illustrated in Figure 1 is the inclusion of a floating-point number (0.6) just after the Stagecoach entity in the "enter" predicate. This number is an iconicity value, which is a real number in

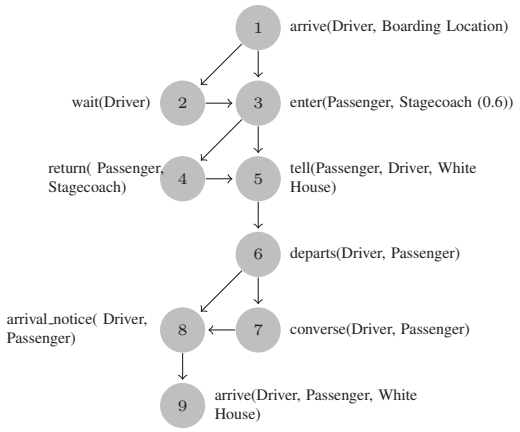

Figure 1: Stagecoach source input script

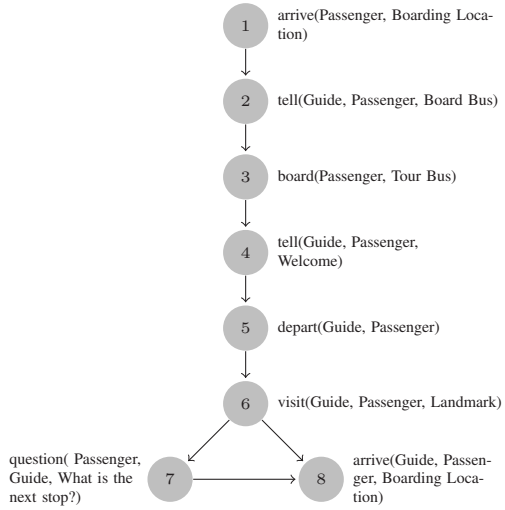

Figure 2: Tour bus target input script

the range [0,1] that indicates the uniqueness of an entity (or predicate) in a script within the context of a script knowledgebase (Magerko, Dohogne, and DeLeon 2011). For example, a stagecoach (in the modern Western world) is commonly used in the context of a horse-drawn trip within a particular locality, and thus is assigned an iconicity value of 0.6. Predicates can also be iconic; for example, a "pitch" action is nearly exclusively associated with baseball, and thus would be assigned a high iconicity value in a baseball script[1].

Our algorithm contains three phases: 1) counterpart mapping, 2) mapping selection, and 3) mapping application. At a high level, our approach combines the path representation mentioned by Veale et al. with the notions of one-to-one mapping and parallel connectivity from structure-mapping theory (Gentner 1983; Veale, O'Donoghue, and Keane 2000). As mentioned above, the blend example contains the stagecoach script as the source input and the tour bus tour script as the target input. In our algorithm, the target script is always the input that is modified in order to generate a script blend (an approach inspired by metaphor).

[1]"Pitch" would also have a high iconicity value in a script for the game of Cricket. This demonstrates the importance of context of a knowledgebase when calculating iconicity values.

Counterpart-mapping Phase

The first algorithm phase, known as *counterpart-mapping*, identifies all paths through the input scripts, given a start node and an end node in each script. Paths are identified using a modified depth-first search that continues to search for paths until all edges in the graph have been traversed. This depth-first search step is guaranteed complete due to the use of DAGs. We attempt to model the notion of "similar activities" as a formalism of structural similarity between a pair of paths in the two input scripts, which retains the temporal ordering of nodes as a form of "higher-order" structure when comparing nodes during the next step.

After all paths have been found, the algorithm establishes counterpart mappings via three node match rules:

1. *Exact-match*: the predicate in the source and target nodes match (using literal equality) and both the order and number of entities matches. Note that this rule is a requirement for both predicate equality and parallel connectivity (from structure-mapping theory). Note that nodes common to both scripts will be mapped, which provides for temporal coherence.
2. *Predicate-match*: a relaxation of the *exact-match* rule where exactly one entity differs. The strictness of exactly one different entity is an effort at ensuring coherence of generated blends. For example, eat(Dog, Dog Food) does not have a predicate-match with eat(Person, Chinese Food) because two arguments differ. However, eat(Dog, Chinese Food) would match with eat(Person, Chinese Food) because exactly one argument differs.
3. *Ordered-entity-match*: similar to the *exact-match* rule, but predicates are allowed to differ. That is, the requirement is for corresponding arguments to match exactly.

Holyoak and Thagard built ACME as an analogical reasoning system. Our approach differs primarily by requiring what Falkenhainer et al. term "parallel connectivity", with a slight relaxation from exact matches for all arguments to exact matches on all arguments except one (in the case of the *ordered-entity-match* rule).

During *counterpart-collection*, every pair of paths (taking one from the source and one from the target to make a pair) is used for pairwise comparison of nodes. This comparison technique enforces temporal ordering, a feature of scripts that must be retained for comprehension. Each node in the source and target is tested for a match using the three rules above. If a match is found between a node n_i in the source input and node n_j in the target input, a candidate counterpart mapping is proposed. An intentional restriction our algorithm places upon further matches is that matches are only possible using nodes that are successors of nodes n_i (e.g., $n_{i+1}, \ldots$) and n_j (e.g., $n_{j+1}, \ldots$). That is, given a mapped pair of nodes (n_x, n_y), the only potential additional matches involve nodes in the source in the set $\{n_{x+1}, \ldots, n_S\}$ and

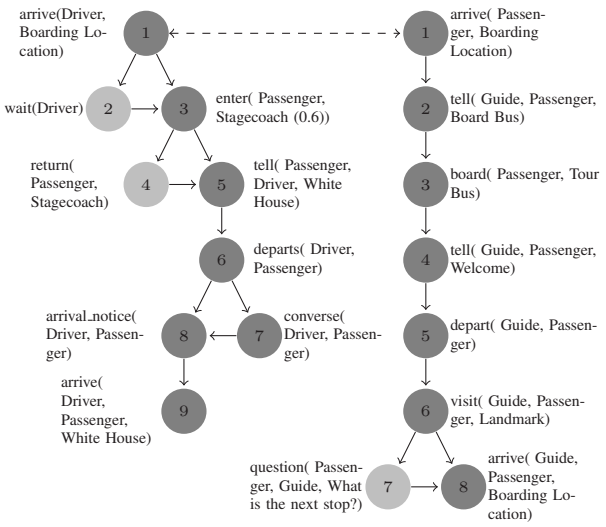

Figure 3: Stagecoach and Tour Bus Counterpart Mappings

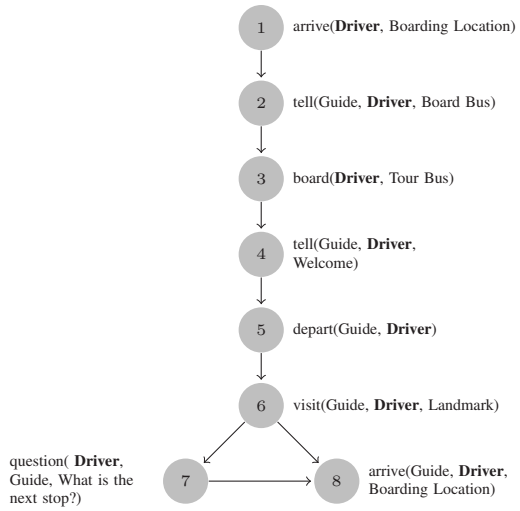

Figure 4: "Stagecoach" + "Tour bus" output blend

nodes in the target in the set $\{n_{y+1}, \ldots, n_T\}$ where S is the number of nodes in the source path in the current path pair and T is the number of nodes in the target path in the current path pair. In this manner, any nodes that match under the constraint of temporality are found for each path. Similar to the approach by Zook et al., we identify attributes common to both inputs and employ a process of attribute substitution during blending. A key difference, however, is the temporal nature of scripts, which requires a significantly different approach to counterpart mapping. An example pairwise path mapping comparison is show in Figure 3. The "active" path through each script is shown in bold.

Figure 3 can be used to illustrate the restriction of temporality used during counterpart mapping. The "arrive" nodes are mapped to each other. Consequently, the only nodes available for further counterpart mapping match tests are nodes *after* the "arrive" nodes.

Mapping-selection Phase

Mapping-selection takes as input the pairs of counterpart mappings from the *counterpart-mapping* phase in order to prioritize potential mappings. Mappings are selected as follows:

1. The first pass through all pairs of mappings identifies iconic mappings. If the mapping involves an iconic node from the target, the mapping is discarded. The reason is that iconicity is a distinguishing characteristic that we want to preserve. For a clear illustration, a pirate is a character typically wearing a pirate's hat, sword, and eye patch. However, if you remove those accessories, the pirate is no longer a pirate, but a relatively uninteresting person. For the same reasons, iconicity in the source is a characteristic that, if possible, we want to transfer to the target. For this reason, if a mapping contains an iconic source node, then we prioritize this mapping for selection.
2. The second pass through the mappings sorts the mappings sets in order of descending set size. For example, if the *counterpart-mapping* phase found 5 mappings for path pair $(path_{S1}, path_{T1})$ and 2 mappings for the path pair $(path_{S2}, path_{T2})$, then these two sets of mappings are sorted in descending order (a set of 5 pairs then a set of 2 pairs). Mappings are used so long as they do not specify previously-mapped nodes, thereby ensuring a one-to-one node mapping.

As mentioned above, our current mapping selection heuristic is to choose mappings in order of descending set size, which is a greedy algorithm. Our motivation for using a greedy selection algorithm is to have a large set of selected mappings at the end of the *mapping-selection* phase. This may prove not to be the best heuristic for mapping selection, which is an area for future work.

Modify-target Phase

The *modify-target* phase uses selected mappings to modify the target input script. The match rules are designed to allow only one difference in a match. The only difference between two nodes matched with the *predicate-match* rule is one entity. Similarly, the only difference between two nodes matched with the *ordered-entity-match* rule is the predicate. Consequently, this phase involves transfer of the difference from the source to the target. The only noteworthy detail is that entity transfer is a global re-mapping of the entity akin to a variable rebind. From figure 3 above, the *predicate-match* rule maps Driver (source script) $\rightarrow$ Passenger (target script). It would be incoherent to map the entity only for that one specific node. Predicate changes, however, are local to the target node specified in the mapping.

Figure 4 illustrates the output. All modifications to the target input script are in bold. The blend script can be interpreted as a training day for a new tour bus driver.

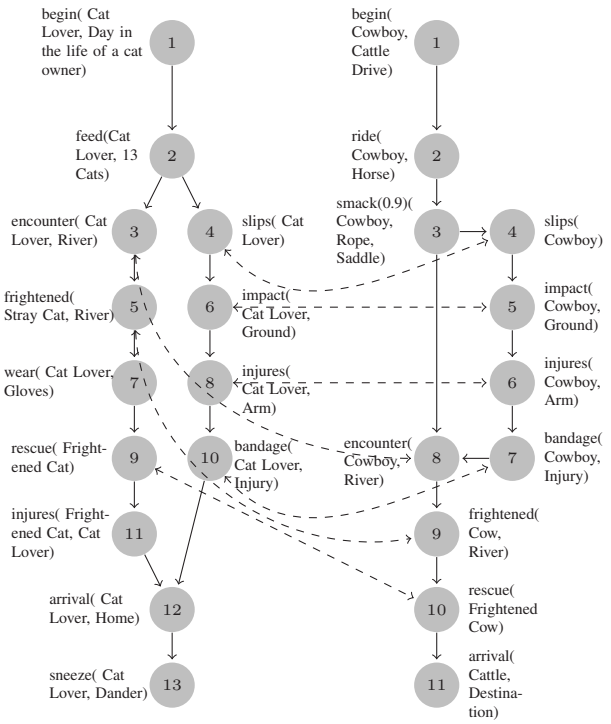

Figure 5: Cat owner and Cattle drive counterpart mappings

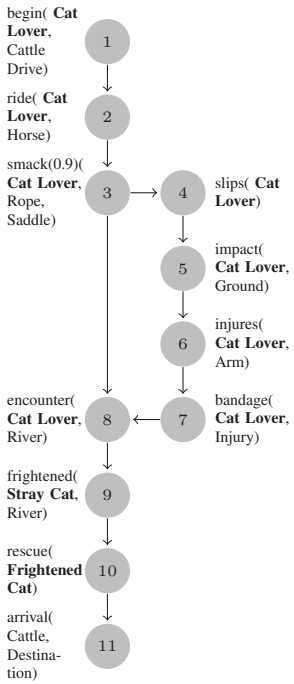

Figure 6: "Cat Lover" + "Cattle Drive" output blend

4 Script Generation Examples

The first example, inspired by TV commercials, is a blend between "A Day in the Life of a Cat Owner" (source input) and "Cattle Drive" (target input). Figure 5 depicts the source input (on the left), the target input (on the right), and selected counterpart mappings, all of which are created due to the *predicate-match* rule. Consequently, all of the changes to the target script to produce the blended script are (global) entity substitutions. Figure 6 depicts the output, a script containing actions and events that could be summarized as "Cat Lover Joins the Cattle Drive". All modifications to the original target input script are in bold.

The second example is a blend between a source script of Derrick Rose, a member of the Chicago Bulls NBA basketball team, charging the basketball hoop and a target script of a bullfight. Figure 7 depicts the source and target inputs and selected counterpart mappings. A key aspects of this blend, which is not explicitly modeled computationally, but which is readily accessible to humans, is the linguistic double-meaning of the word "bull", referring to both a bull in a bullfight and Derrick Rose as a Chicago Bull. Both meanings appear in the source and target scripts and allow the algorithm to establish the "creative" counterpart mappings depicted in Figure 7. Figure 8 illustrates the script blend, with predicate and entity substitutions in bold.

5 Discussion

In this section, we discuss a few noteworthy details and salient characteristics of our algorithm and approach.

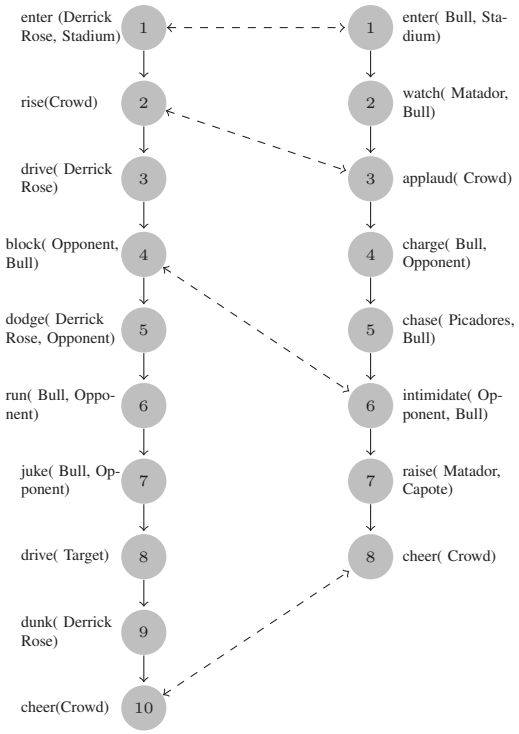

Figure 7: Derrick Rose and Bullfight counterpart mappings

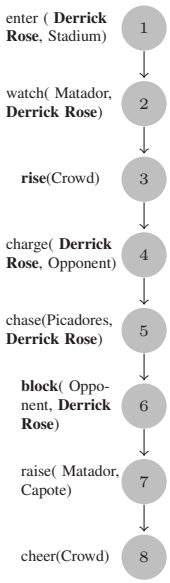

Figure 8: "Derrick Rose" + bullfight output blend

Our algorithm is currently designed to perform predicate and entity substitutions. We chose these operations in order to produce coherent, executable blends. As mentioned above, our algorithm is designed for use by an intelligent agent in narrative contexts like pretend play. The algorithm assumes that the input scripts are both valid and executable, and it is vital that the algorithm produces executable scripts as well.

A predicate substitution occurs when a mapping is created as a result of an *ordered-entity-match*. In this case, all of the arguments match. Moreover, the predicate is valid in the source script for the matched arguments. As a result, there is a "high" likelihood that the transfer of the predicate from the source to the target will be both coherent and executable. However, there is no guarantee, perhaps due to a mismatch of context or a lack of ability to perform the action in the blended script (perhaps due to missing pre-conditions for the action to occur that existed in the source script, but do not exist in the target script).

Narratives are comprised of both causality and temporality, which led us to consider how temporality and action/event ordering influences our perception of similarity between activities (Rimmon-Kenan 2002). Because our scripts explicitly specify temporality, we ensure that counterpart mappings account for node ordering. This requirement, in turn, helps to ensure that the algorithm produces coherent, executable scripts by avoiding the introduction of violations of pre-conditions that may result from reordered nodes.

Limitations

As discussed above, our algorithm uses two features of scripts, temporality and iconicity, during blending. These surface-level features do not provide us with the semantic information required to maintain causality, which is a strong requirement for coherence of a blended script. Guaranteeing causality, and thus coherence, requires a vast amount of general and specific knowledge that we do not currently use in our approach. As a result, our algorithm can produce incoherent blended scripts. Additionally, our algorithm produces blends only for two inputs scripts (and not n-ary inputs).

Another limitation of our current approach is the use of a literal (syntactic) match when testing for equality between the predicates or entities in two nodes. For a given knowledge base, this could be a significant limitation to identifying existing counterpart mappings. This could, in theory, be resolved in a reasonably straightforward manner by using another equality test. For example, perhaps a minimum similarity threshold for two entities, given a (presumably) common prototype.

Future Work

This paper presents first steps toward a script generation algorithm. We are currently undertaking an effort to acquire a script knowledge base in order to evaluate our approach. Concurrent with that effort, we have identified a few possible extensions to our algorithm to ensure it is generally applicable.

First, our current algorithm uses only two operations to produce a blend: 1) entity substitution and 2) predicate substitution. These substitutions do not provide us with an ability to augment (modify) the structure of the target graph. For example, it may make sense to "graft" nodes from the source script into the target script in the case where there is an established counterpart mapping between nodes. This would provide a significant additional capability of executing actions originally specified in the source script that have no counterpart in the target.

Second, we intentionally designed the algorithm to produce one blended joint activity for execution by an intelligent agent. However, SME and other systems regularly produce multiple blends with a relevant notion of "score" that could prove useful. As noted above, there is no guarantee that a given blend is executable. Thus, a score metric could be devised that measures executability using the play context.

Third, our algorithm may benefit from using contextual information, perhaps by employing a commonsense knowledge base such as ConceptNet (Liu and Singh 2004). Consider a scenario where a "Cooking" source script contains the predicate cook(Person, Food) and a "Picnic" target script contains the predicate eat(Person, Food). A counterpart mapping can be established between the two nodes according to the *ordered-entity-match* rule, resulting in a predicate substitution in the "Picnic script" of "eat" $\rightarrow$ "cook", which is incoherent because cooking cannot occur during a picnic, due to a lack of a heat source[2]. A similar problem occurs for entity substitution. An additional post-processing step could be used to identify, and subsequently filter, "inappropriate" blends based on context, which could be knowledge intensive.

[2]We admit that there could be exceptions, such as a picnic with a fire, but a canonical picnic does not involve a fire.

References

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

Fauconnier, G., and Turner, M. 1998. Conceptual integration networks. *Cognitive science* 22(2):133–187.

Fauconnier, G., and Turner, M. 2003. *The way we think: Conceptual blending and the mind's hidden complexities*. Basic Books (AZ).

Forbus, K. D.; Gentner, D.; and Law, K. 1995. MAC/FAC: a model of similarity-based retrieval. *Cognitive Science* 19(2):141–205.

Gentner, D. 1983. Structure-mapping: A theoretical framework for analogy. *Cognitive science* 7(2):155–70.

Goguen, J., and Harrell, F. 2004. Foundations for active multimedia narrative: Semiotic spaces and structural blending. *Interaction Studies: Social Behaviour and Communication in Biological and Artificial Systems*.

Harrell, D. F. 2005. Shades of computational evocation and meaning: The GRIOT system and improvisational poetry generation. In *Proceedings, Sixth Digital Arts and Culture Conference*, 133–143.

Holyoak, K. J., and Thagard, P. 1989. Analogical mapping by constraint satisfaction. *Cognitive science* 13(3):295–355.

Li, B.; Zook, A.; Davis, N.; and Riedl, M. O. 2012. Goal-driven conceptual blending: A computational approach for creativity. In *International Conference on Computational Creativity*, 10.

Liu, H., and Singh, P. 2004. ConceptNet – a practical commonsense reasoning tool-kit. *BT Technology Journal* 22(4):211–226.

Magerko, B.; Manzoul, W.; Riedl, M.; Baumer, A.; Fuller, D.; Luther, K.; and Pearce, C. 2009. An empirical study of cognition and theatrical improvisation. In *Proceedings of the seventh ACM conference on Creativity and cognition*, 117–126.

Magerko, B.; Dohogne, P.; and DeLeon, C. 2011. Employing fuzzy concepts for digital improvisational theatre. In *Proceedings of the Seventh Annual International Artificial Intelligence and Interactive Digital Entertainment Conference*.

Moran, S.; John-Steiner, V.; and Sawyer, R. K. 2003. Creativity in the making. *Creativity and development* 61–90.

Rimmon-Kenan, S. 2002. Narrative fiction: Contemporary poetics (new accents).

Schank, R. C., and Abelson, R. P. 1977. Scripts, plans, goals and understanding: An inquiry into human knowledge structures.

Veale, T.; O'Donoghue, D.; and Keane, M. T. 2000. Computation and blending. *Cognitive Linguistics* 11(3/4):253–282.

Zook, A.; Magerko, B.; and Riedl, M. 2011. Formally modeling pretend object play. In *Proceedings of the 8th ACM conference on Creativity and cognition*, C&C '11, 147–156. New York, NY, USA: ACM.