Intelligent Narrative Technologies: Papers from the 2012 AIIDE Workshop
AAAI Technical Report WS-12-14

Generating Narrative Action Schemas for Suspense

Spyridon Giannatos, Yun-Gyung Cheong, Mark J. Nelson, and Georgios N. Yannakakis
Center for Computer Games Research
IT University of Copenhagen
Copenhagen, Denmark
{spgi,yugc,mjas,yannakakis}@itu.dk

Abstract

A bottleneck in interactive storytelling is the authorial burden of writing narrative units, and connecting them to the interactive narrative structure. To address this problem, we present a hybrid approach that combines AI planning and evolutionary optimization in order to generated new plan operators representing possible story actions, within the framework of a planning-based interactive narrative system. We focus our work on inventing plan operators that are useful for contributing to suspenseful interactive stories, using suspense metrics that have been proposed in the literature. We devise an encoding scheme for converting a plan operator into a genetic-algorithm chromosome and vice versa, respecting constraints that are needed for an operator to be well-formed. We discuss the performance of the system, and several examples from preliminary experiments carried out to evaluate the evolved operators.

Introduction

Interactive storytelling systems often have a flavour of sequencing content, or guiding the sequencing of content. The space of possible stories that a particular system supports can be represented in several ways: for example, as a graph of important plot points and sequences through them (Nelson et al. 2006), or, in a planning framework, as story state and operators that modify story state (Young 1999). A drama manager then ensures that traversals through this story space meet certain properties, to maintain narrative coherence and any other, more specific, properties the author may wish the stories to have. A long-term goal of many interactive storytelling systems is to turn this content-sequencing approach into a more generative approach, where new content is created as needed. This problem is related to a larger AI research topic—automated knowledge learning.

In this paper, we explore adding generativity to an interactive storytelling system based on a planning representation. As is common in such representations, the story world, and interaction within it, are modelled by: 1) predicates, and 2) action operators that refer to those predicates in their pre- and post-conditions. The predicates specify an ontology of the story world, and the actions are story events that can happen, defined within that ontology. Here, we focus on inventing new planning operators, defined within an existing story-world ontology (in other words, we do not invent predicates). This makes the scope of the problem more tractable, since the generation problem is kept within a background setting that the author has already ensured makes sense. However, it can still add a significant generative component to interactive stories, because the defined operators, that is, the story's actions and events, are a key to interesting interactive stories. To produce sensible stories, these operators need to cover a space less than all possible operators that *could* be defined with a set of predicates, but we hypothesize that there should be more operators than are typically hand-authored for a particular domain.

To further focus our research, in this work we aim to invent story actions for one kind of storytelling: telling suspenseful stories. Therefore new operators are judged by how useful they are for creating suspense in a given context, which is a particularly interesting problem because suspense is not a property of a single event in a story, but of how the events fit together; and therefore there may be very different solutions in different story contexts. We generate operators via FI2POP (Kimbrough et al. 2008), an evolutionary algorithm that performs optimization within a set of constraints. The constraints are that the generated operators must be well-formed, with preconditions and postconditions that share some parameter types. Among those well-formed operators, the metric being optimized is one based on theories of storytelling suspense.

On a tripartite narrative generation model which consist of *fabula*, *sjuzhet*, and discourse layers (Cheong 2007)—where *fabula* denotes a complete story containing all the events happening in the story world, *sjuzhet* contains only those events and their orderings that are presented to the audience, and discourse is the final product (e.g., text, film, animation) delivered to the audience—our system contributes to the fabula generation layer by the provision of varied operators as bulding blocks for building a *fabula*. Although it is our intention to implement a fully automated system that creates, evaluates, and annotates semantic meanings to the newly generated narrative units, a practical application of the current work would be to suggest plausible operators for story authors for authoring systems.

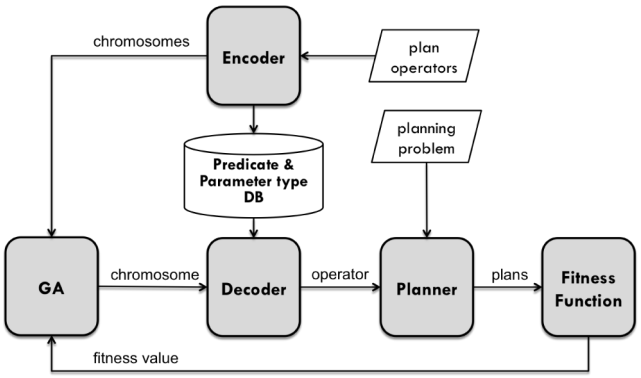


Figure 1: The proposed framework

Our approach

The goal of our system is to generate plan operators as narrative units for constructing a story represented as a plan structure which consists of story actions and their causal relationships and temporal constraints. Planning technology has been widely used in interactive storytelling systems (Lebowitz 1983) due to its capability of generating interactive stories (Trabasso and Sperry 1985; Cavazza, Charles, and Mead 2002) via re-planning and its data structure and algorithm can account for the reader's story comprehension process (Christian and Young 2003). More on the representations for interactive storytelling can be found in (Magerko 2005; Roberts 2011).

Figure 1 shows our framework for generating narrative actions. It consists of a genetic algorithm (GA) component, and encoder and decoder, a planner, and a fitness function. The system additionally contains data structures and a predicate database.

In the initialization step, the encoder creates the *predicate database*, consisting of a list of all the predicates (i.e., logical sentences) in a planning domain that is given as input. Each of these predicates can be indexed by *predicate ID*. The system also creates the *parameter type database*, which is a list of the parameter types that exist in the planning domain.

The system repeats the following steps until it reaches convergence or a pre-defined number of generations:

I The GA component generates a set of genotypes through an evolutionary process guided by the fitness function. These genotypes serve as input to the decoder.

II The decoder converts a chromosome from the GA into a plan operator structure, consisting of preconditions and effects.

III When the new operator is received from the decoder, the planner generates solutions to the planning problem using the set of initial plan operators augmented with the newly generated operator.

IV The fitness function takes the solutions as input and returns a numeric value representing the new operator's utility to the GA component.

Figure 2: A predicate block which consists of the predicate ID as an integer and a binary value representing whether the predicate is true or false.

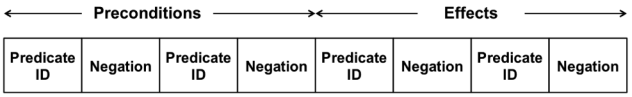


Figure 3: Chromosome representation, as an array of predicate blocks.

Encoding scheme

Evolutionary algorithms are a standard technique for optimization problems, in which the solutions are often represented as binary or numeric values. On the other hand, AI planning is an example of a symbolic approach in which each condition in the preconditions/effects of a plan operator relates to a particular state and object in a story world. Therefore, it is challenging to give semantic meaning to the chromosome, which can be a random combination in the worst case. This section discusses how we encode plan operators in a way that a GA can comprehend and vice versa. A plan operator is a tuple of three elements—(name, a set of preconditions, a set of effects)—where preconditions and effects reference the domain's predicates. We begin with the description of predicates since they are the basic building blocks.

Predicate representation To bridge the two different representations (numeric values and logical sentences), we employ a predicate database, where each tuple in the DB contains an index which can be used as both a gene value, and a way of accessing the corresponding predicate. In our approach, a predicate is encoded as a gene which contains an integer denoting predicate ID and a binary value denoting whether the predicate is true or false in this instance (see Figure 2. Therefore, a plan operator is represented as a series of predicate blocks, as illustrated in Figure 3. In this paper, we use a fixed-length chromosome, and therefore the number of predicates representing a plan operator is also fixed, at two conditions and two effects. The length of chromosome was chosen through preliminary experiments.

Parameter binding problem One of the main challenges in our framework is parameter assignment. Consider the following example to illustrate the issue. Imagine a plan operator containing the following preconditions and effects.

Preconditions: $at(?x1)$ AND $\neg has(?x2)$
Effects: $has(?x3)$

This operator has three variables ($?x1$, $?x2$, $?x3$). When these variables are bound to particular parameter types, the plan operator may produce different meanings. For instance, if the predicate at binds a $place$ type value to $?x1$ and the predicate has binds an $item$ type value, we can imagine

that this operator may mean PICK-UP under the condition of ($?x2 = ?x3$). However, if the parameter type for $?x2$ is different from the one for $?x3$, the PICK-UP action is not the best description of the operator.

In other words, constraining the parameter types of variables influences semantic meaning. A full solution to the parameter assignment problem is beyond our current research scope, but as a simple solution, we view this as a constrained optimization problem, in which parameter types in preconditions are constrained to match the types in effects. In order to meet this requirement, we use the FI2POP (Feasible-Infeasible Two-Population) genetic algorithm for the GA component.

FI2POP GA

Unlike most evolutionary algorithms, which deal only with valid solutions, FI2POP (Kimbrough et al. 2008) is a constrained optimization algorithm that maintains feasible and infeasible populations throughout the evolutionary process. The feasible population contains only feasible individuals (those that satisfy the constraints) and the infeasible population contains only infeasible individuals (those that don't). Every individual is tested for feasibility and placed in the appropriate population. Individuals in the feasible population are selected by fitness with respect only to their objective function values. Individuals in the infeasible population are selected by fitness with respect only to function of their constraint violations. The infeasible population tends to probe the neighbourhood of the boundary of the feasible region. Children tend to resemble their parents, so feasible children of infeasible parents tend to be close to each other. Because the infeasible parents were selected without regard to their objective function, crossover and mutation will tend to create feasible solutions that presumably are different from those already in the main population. These solutions either succeed or fail, but in either case they contribute to an exploration of the feasible region. Thus, if the optimal solution is on or near the boundary, it is expected that the infeasible population would contribute to finding it.

Measuring the coherence of an evolved operator relies on domain knowledge. We use a *parameter type database* derived from co-occurrences of types in the original domain. If a pair of parameter types does not appear together in any of the operators in the original domain, the coherence of the operator is scored more poorly.

The coherence test is done by the means of penalty function or *constraint violation function*, which is used by the FI2POP GA. This function accepts the chromosome and returns the number of violations of coherence. This function starts by enumerating all possible combinations of predicates of our evolved operator. Then, for each pair, the function loops through all operators of the original domain. If there is an operator that contains the pair of predicates in question, then no violations exist for this pair and we continue with the next pair of predicates. In the case where there is no operator containing the pair of predicates in question, we increase the violations by one. The threshold for *constraint violation function* is set 3 in this study. It means that a plan operator which has scored over 3 in constraint violation falls in the infeasible population.

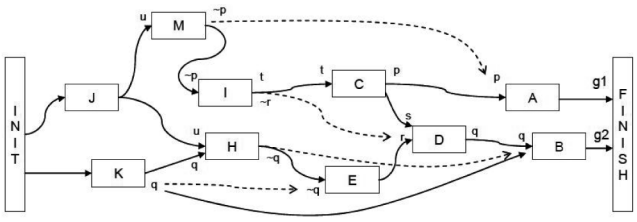


Figure 4: A plan structure containing supporting and threatening links (Cheong 2007). Actions (with their preconditions on the left and effects on the right) are represented by boxes. Solid arrows denote supporting links, meaning that an action's effect is unified with a precondition of another action. Dotted arrows are threatening links, which means that an action's effect is a negation of another action's precondition. Please note that these terms are used unconventionally. In conventional planning, a plan is considered complete when there are no threats to causal relationship. The threatening links in this paper are temporary threats which are resolved eventually. For instance, the threatening link connecting the step *M* to the step *A* via the negation of *p* is resolved by the subsequent step *C* that establishes the precondition *p*. Similarly, the supporting link connecting the step *K* to the step *B* via the condition *q* has been threatened by the step *H*'s effect negating *q* and re-established by the step *D*.

Planner

Our approach uses a C# version of the Longbow planner, which generates hierarchical, partial-order causal link plan structures (Young, Pollack, and Moore 1994). When the decoder passes a new plan operator to the planner, it builds a plan space which is composed of nodes representing partial and complete plans. The planner finds the solution for a given planning problem that achieves the goal state, starting from the initial state by repairing the flaws in the partial plans—open preconditions, threatened causal links, and abstract steps which have not been decomposed into primitive actions. This refinement search process continues until no partial plans are left, or the number of nodes in the space exceeds a pre-defined search limit. A plan structure consists of plan steps, binding constraints, temporal ordering information, causal links, and decompositional links, as illustrated in Figure 4. When the search process terminates, the planner returns all the solutions to the fitness function. Note that the planner considers the new plan operator as well as the initially given plan library.

Fitness function

In evolutionary algorithms, fitness functions take a genotype as input and return a numeric value representing the optimality of the provided genotype-solution. In our approach, the utility of a solution is an estimate of its contribution to generating suspenseful stories. While there are various techniques for creating suspense in stories, our fitness function

focuses on one primary condition for suspense: the reader feels suspense when anticipating an undesirable outcome for the protagonist (Gerrig and Bernardo 1994).

In order to gauge the potential suspense that the new operator can contribute, we rely on a heuristic function calculating *the potential suspense of an action in a plan* in the approach to story structure generation for suspense (Cheong 2007). Simply put, the heuristic function in computes the potential suspense of an action based on the number of supporting links and threatening links (see Figure 4 for definitions).

Potential suspense calculation It is expected that the more the threatening links hold, the higher the suspense is. On the other hand, the more the supporting links hold, the lower the suspense. Hence, we calculate the potential suspense level of a story plot p through the function:

$$Susp_{plot}(p) = t(p)/s(p) \quad (1)$$

where $t(p)$ and $s(p)$ are functions that calculate the weighted sum of threatening and supporting links respectively.

Since there exist multiple solutions, the system needs to compute the potential suspense of each plan in all the solutions. Therefore, our *Potential Suspense* fitness function is given by the equation:

$$Suspense(o, s) = \sum_{p \in Plots(o,s)} Susp_{plot}(p) \quad (2)$$

where $Plots(o, s)$ returns all complete plans in the set of solutions s that contain actions instantiated from the new operator o. The *potential suspense* fitness function is domain-independent, which give us the advantage of using the generic framework for various domains.

Evaluation

Experimental setting

We demonstrate this approach in a simple narrative domain we created (we call it *Sykes' Mission*). In this story, an actress named Agatha is in danger of being assassinated, but she isn't aware of this fact. Sykes plots to kill Agatha, and plans to obtain a bomb from Kent (who has a bomb). One solution to this problem that the planner generate is to let Sykes get a loan in order to buy a bomb from Ken. Then Sykes carries the bomb to the theatre on the date when Agatha is visiting. He installs the bomb, sets the timer, and finally switches it on. These actions establish the preconditions of the operator *explode*, which is the final action achieving the goal of ¬ *alive(Agatha)*.

The following GA setting is carefully chosen through preliminary experiments. The termination condition of our GA application is set to 50 generations. The population size is 40. In the crossover step of the GA we use *Uniform Crossover* with crossover rate 80% and in the mutation step we change we gene value randomly with mutation rate 1%. Selection via elitism is an important aspect of the GA, as it helps to preserve parents with good fitness values. In our experiments, the elitism rate is set to 5%. The entire process is then executed ten times for each domain.

Table 1: This table presents the features of the best evolved operator (*Operator A*) and the worst (*Operator B*) that our framework managed to produce using the original domain *Sykes Mission*.

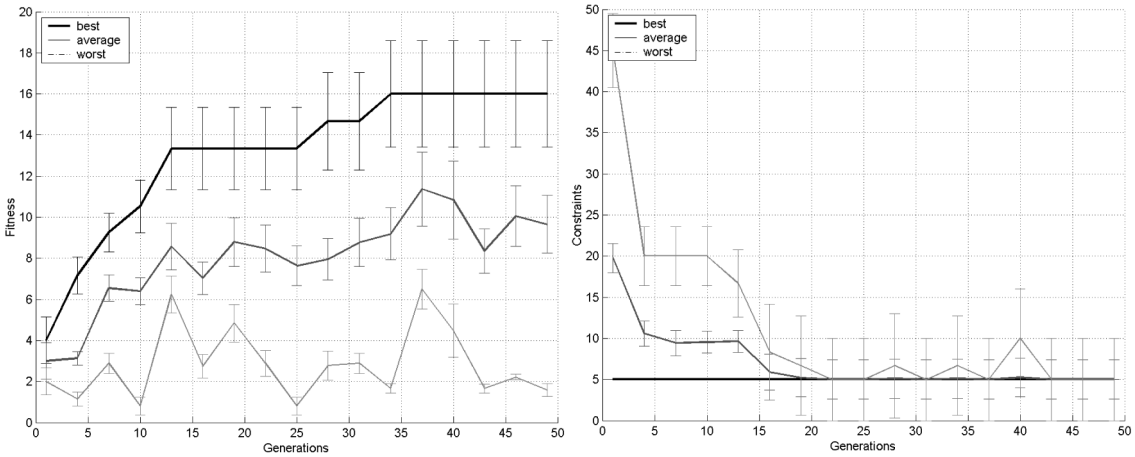
Features	Operator A	Operator B
Genes	8	8
Fitness value	16	1
Parameters	?bomb ?person	?person ?bomb ?place
Constraints	isabomb(?bomb) isaperson(?person)	isaperson(?person) isabomb(?bomb) isaplace(?place)
Preconditions	has(?person, ?bomb) ¬ installed(?bomb)	¬ at(?bomb, ?place) has(?person, ?bomb)
Effects	installed(?bomb) ¬ on(?bomb	on(?bomb) installed(?bomb)

Results

We performed a pilot study demonstrating this method in operation. First, we show the graphs of the GA evolution and its corresponding infeasible population's growth. We then describe the best and worst operator we obtained, and show how the best one maintains coherence and increases suspense inside a plan instance. Figure 5 presents the results using the *Potential Suspense* fitness function for a story. These charts show the progress of the best, the average, and the worst evolved operators. While the fitness for the best operator reaches convergence, the other two showed some fluctuation.

We obtained the best and worst evolved operators from the experiment. Table 1 shows the important features of these operators: the chromosome length, the fitness value, the parameters and their constraints, the preconditions and the effects.

Best operator *Operator A* in the table is the best evolved operator that our framework managed to produce for the *Sykes' Mission* story. As the preconditions state, this operator can be executed only when a person has a bomb and the bomb is not yet installed. After executing this action, the bomb is installed, but the switch is turned off (even though it might be on before). The original domain contains an operator which is called *installbomb* and has the effect of installing the bomb. This new operator is quite similar to the *installbomb* that exists in the original plan operators. The difference is that the new operator keeps the bomb off. We interpreted the meaning in two ways. First, this operator can serve as a version of *installbomb* which requires another action such as *switch-on* for the bomb to operate. An alternative interpretation is to treat the new operator as abstract which combines two actions: 1) installation of a bomb, followed by 2) an accidental damage to turn it off. Giving a se-



(a) Feasible population's progress (b) Infeasible population's progress

Figure 5: Evolution of the *Potential Suspense* fitness function tested across the *Sykes Mission* planning domain, using chromosomes of 8 genes. The figure on the left shows the progress of the feasible population with 8 genes (4 predicates), while the figure on the right shows the progress of the infeasible population. Figures depict average values of 10 runs and corresponding 95% confidence interval bars every 3 generations.

mantic meaning to the operator is very important when we try to convey that the new operator happened in a story to the story consumer in a form of medium. For instance, an action instantiated from the operator can be described as text, "Sykes installed the bomb and it was turned off", or can be visualized showing the corresponding action. As described above, the system can take the name of a similar operator in the current library. Another way is to describe/show the state before the execution of the operation and the state after the execution, without realizing the action behavior. The investigation of effective presentation of this nameless operator would fall in the discourse layer and is beyond our scope.

Worst operator *Operator B* (Table 1) was the worst evolved operator for comparison with the best operator. The worst operator is the evolved operator that has the lowest positive fitness function and opposed to the best operator, *Operator B* has higher number of supporting links in the plan instances—i.e. decreased suspense. The preconditions of this operator are: 1) a bomb is not at $?place$, and 2) a $?person$ has a bomb, which is true in the initial state when $?person$ is bound with Kent and $?place$ is bound with theatre. After executing *Operator B*, the bomb is suddenly installed and the switch is on. As a result, augmenting this operator in the plan library enables the planner to generate a very short story which does not necessarily involve Sykes in assassination, which in return gives the reader little chance to experience suspense.

Related work and background

Learning primitive actions in narratives relates to work beyond interactive narrative components in particular. In existing research (Li et al. 2012), sociocultural knowledge (e.g., going to a fast food restaurant, taking a date to a movie) was learnt from crowd-sourced narrative examples. The goal of their approach is to learn a) actions through unsupervised clustering and b) the causal, temporal relations between two events based on probabilistic methods. The difference between their approach and ours is two-fold. First, unlike their approach the generation of actions in this research is guided by a fitness function. Second, the action script in our research is a set of preconditions/effects, while their approach is a precedence relationship. We propose to use their research as a component to name the operator. For instance, imagine a new operator containing the precondition of *having paid for food* and the effect of *carrying food*. The approach in (Li et al. 2012) may suggest actions that can occur between *pay for food* and *eat food* as candidate operator names. If nothing matches the conditions, it is likely that the new operator does not exist in real-life situations. However, we can still consider the operator in a fantasy world.

In the previous work (Giannatos et al. 2011), we have presented a system which can create interactive narrative elements from a direct acyclic graph representing plot points. In this approach, a plot point contains a set of semantic features such as location, thought, and motivation along with temporal ordering constraints in relation to other plot points in the story graph. A standard GA generates a new plot point guided by fitness functions which estimate different types of player's experience based on the location flow, thought flow, and motivation. The new plot point is integrated into the original plot point graph through the incoming arcs from and outgoing arcs to other plot points which represent the temporal ordering relationship. While they have shown the fasibility of combining symbolic and evolutionary appraoches in generating plot elements, the limitation of this approach lies in the lack of semantic information contained in the generated plot point, which makes it difficult to give meaning to the new plot point beyond its formal location in the plot

graph. The approach presented in this paper is an attempt to address that problem by using a planning-based representation, as those structures contain logical sentences representing pre-conditions and post-conditions of an action.

Conclusions and future work

We present a framework that learns actions for generating stories for suspense guided by story evaluation functions. The contributions this work made are 1) to design a system architecture combining an AI planner and evolutionary algorithm, 2) to devise an encoding scheme that translates a plan operator into a chromosome and vice versa, and 3) to devise a fitness function estimating the contribution of a plan operator in generating suspenseful narratives when added to an existing plan library. We also discovered that parameter types bound with preconditions and effects can determine the operator's semantic meaning and therefore need special attention. We have carried out a pilot study which showed a promising result where an evolved operator with highest fitness value seems to be more meaningful than a one with lowest fitness. However, the direct use of the new operator requires a decision in translating the combination of its preconditions and effects into an actionable behavior. Therefore, an application of the work presented in this paper would be to suggest plausible operators for the human domain expert for interactive story authoring tools. To develop a fully automated system without human intervention, we will be investigating the cognitive process of granting semantic meanings to the operator via crowdsourcing approaches.

In the future, we plan to improve the fitness function for better performance and formal evaluation. For instance, the current function measuring the potential suspense of a plan (equation 1) makes it difficult to gauge the maximum and minimum fitness value. Likewise, the equation 2 can be enhanced by taking the average of potential suspense of plans rather than summation. It would be also possible to consider a complicated system that rates an operator's contribution to story generation as a fitness function in our framework. For instance, the suspense detection system Dramatis (O'Neill and Riedl 2011) can predict a reader's suspense ratings at specific moments in a story by taking into account her cognitive capacity in estimating the likelihood of the protagonist's successful plans under time constraints.

References

Cavazza, M.; Charles, F.; and Mead, S. J. 2002. Character-based interactive storytelling. *IEEE Intelligent Systems* 17:17–24.

Cheong, Y.-G. 2007. *A Computational Model of Narrative Generation for Suspense*. Ph.D. Dissertation, North Carolina State University.

Christian, D. B., and Young, R. M. 2003. Comparing cognitive and computational models of narrative structure. In *Proceedings of the 19th National Conference on Artificial Intelligence*, 385–390.

Gerrig, R. J., and Bernardo, A. B. I. 1994. Readers as problem-solvers in the experience of suspense. *Poetics* 22(6):459 – 472.

Giannatos, S.; Nelson, M. J.; Cheong, Y.-G.; and Yannakakis, G. N. 2011. Suggesting new plot elements for an interactive story. In *Proceedings of the 4th Workshop on Intelligent Narrative Technologies, AIIDE*. AAAI Press.

Kimbrough, S.; Koehler, G.; Lu, M.; and Wood, D. 2008. On a feasible-infeasible two-population genetic algorithm for constrained optimization: Distance tracing and no free lunch. *European Journal of Operations Research* 190(2):310–327.

Lebowitz, M. 1983. Creating a story-telling universe. In *Proceedings of the Eighth international joint conference on Artificial intelligence - Volume 1*, 63–65. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Li, B.; Appling, D. S.; Lee-Urban, S.; and Riedl, M. O. 2012. Learning sociocultural knowledge via crowdsourced examples. In *Proceedings of the 4th AAAI Workshop on Human Computation*.

Magerko, B. 2005. Story representation and interactive drama. In *Proceedings of the First Conference on Artificial Intelligence and Interactive Digital Entertainment*, 87–92.

Nelson, M. J.; Mateas, M.; Roberts, D. L.; and Isbell, Jr., C. L. 2006. Declarative optimization-based drama management in interactive fiction. *IEEE Computer Graphics & Applications* 26(3):30–39.

O'Neill, B., and Riedl, M. 2011. Toward a computational framework of suspense and dramatic arc. In *Proceedings of the 4th International Conference on Affective Computing and Intelligent Interaction*, 246–255. Springer-Verlag Berlin, Heidelberg.

Roberts, D. L. 2011. Seven design challenges for fully-realized experience management. In *Intelligent Narrative Technologies 4, AIIDE*.

Trabasso, T., and Sperry, L. L. 1985. Causal relatedness and importance of story events. *Journal of Memory and Language* 24(5):595 – 611.

Young, R. M.; Pollack, M. E.; and Moore, J. D. 1994. Decomposition and causality in partial order planning. In *Proceedings of the 2nd International Conference on Artificial Intelligence and Planning Systems (AIPS)*, 188–193.

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

Acknowledgments

This work has been supported in part by the EU FP7 ICT project SIREN (project no: 258453). We thank Arnav Jhala at UC Santa Cruz, and Antonios Liapis and Julian Togelius at IT University of Copenhagen for the discussion.