Argument-based Case Revision in CBR for Story Generation

Santiago Ontañón[1], Enric Plaza[2] and Jichen Zhu[1]

[1] Drexel University
Philadelphia, PA, USA 19104
santi@cs.drexel.edu, jichen.zhu@drexel.edu
[2] IIIA, Artificial Intelligence Research Institute
CSIC, Spanish Council for Scientific Research
Campus UAB, 08193 Bellaterra, Catalonia (Spain)
enric@iiia.csic.es

Abstract. This paper presents a new approach to case revision in case-based reasoning based on the idea of argumentation. Previous work on case reuse has proposed the use of operations such as case amalgamation (or merging), which generate solutions by combining information coming from different cases. Such approaches are often based on exploring the search space of possible combinations looking for a solution that maximizes a certain criteria. We show how Revise can be performed by arguments attacking specific parts of a case produced by Reuse, and how they can guide and prevent repeating pitfalls in future cases. The proposed approach is evaluated in the task of automatic story generation.

1 Introduction

Case-based reasoning systems are based on the hypothesis that "similar problems have similar solutions", and thus new problems are solved by reusing or adapting solutions of past problems. However, how to reuse or adapt past solutions to new problems, and how to revise these solutions are some of the least understood problems in case-based reasoning. There are multiple open problems such as what knowledge is required for adaptation and how to acquire it [20], the relation between solution reuse and case retrieval [17], and solution revision [10]. This paper builds upon previous work on search-based reuse in case-based reasoning, and specifically on approaches based on amalgam or merge operators [12, 3], where a solution to a given problem is generating by amalgamating the problem with one or more retrieved cases.

Specifically, in this paper we focus on the following problem: search-based approaches to case reuse employ some sort of search mechanism over the space of solutions trying to either maximize or satisfy some evaluation function, that hopefully captures the quality of the proposed solution. However, in some domains, such as automated story generation [6] (which we used as our application domain), defining an evaluation function that captures the quality of a solution is a very hard problem. For this reason, we propose a new case revision

approach that integrates argumentation into the case reuse process. Each time the case reuse process proposes a solution, this is evaluated against a collection of arguments that may attack the solution, forcing the case reuse to search for alternative solutions. We claim that rather than capture how "good" a story is, it is easier to define a collection of arguments that attack certain negative aspects of the story. These arguments can be kept for future episodes, to prevent generating stories that suffer from the same problems.

The remainder of this paper is organized as follows. We first introduce some background on amalgam-based case reuse, and on argumentation. After that we present our motivating domain: automatic story generation. Argument-based revision is then presented, followed by an experimental evaluation. The paper closes with conclusions and directions for future work.

2 Background

Stories (cases) are represented in the formalism of feature terms [1], and case reuse is implemented as an amalgam of two feature terms: a source term and a target term. We will briefly introduce here the basic notions of feature terms and amalgamation; for more a detailed explanation see [13].

Feature terms are defined by their *signature*: $\Sigma = \langle \mathcal{S}, \mathcal{F}, \leq, \mathcal{V} \rangle$. $\mathcal{S}$ is a finite set of sort symbols, including $\bot$ representing the most general sort ("any"), and $\top$ representing the most specific sort ("none"). $\leq$ is an order relation inducing a single inheritance hierarchy in $\mathcal{S}$, where $s \leq s'$ means s is more general than or equal to s', for any $s, s' \in \mathcal{S}$ ("any" is more general than any s which, in turn, is more general than "none"). $\mathcal{F}$ is a set of feature symbols, and $\mathcal{V}$ is a set of variable names. We define a feature term ψ as: $\psi ::= X : s \ [f_1 \doteq \Psi_1, \cdots, f_n \doteq \Psi_n]$, where ψ points to the *root* variable X (that we will note as $root(\psi)$) of sort s; $X \in \mathcal{V}$, $s \in \mathcal{S}$, $f_i \in \mathcal{F}$, and Ψ_i is either a variable $Y \in \mathcal{V}$, or a set of variables $\{X_1, ..., X_m\}$. The set of variables present in a term ψ is noted $Var(\psi)$; for instance, the term shown in Fig. 2 has 18 variables, one for each node.

The basic relation over feature terms is *subsumption* ($\sqsubseteq$), i.e. given two terms ψ_1 and ψ_2 we say $\psi_1 \sqsubseteq \psi_2$ (ψ_1 subsumes ψ_2) when ψ_1 is a generalization of ψ_2, or dually ψ_2 is a specialization of ψ_1. Subsumption generates a total mapping m: $Var(\psi_1) \rightarrow Var(\psi_2)$ satisfying certain conditions such as $m(root(\psi_1)) = root(\psi_2)$, or if $X.f = Y$ then $m(X).f = m(Y)$ (formal definition in [13]).

The *unification* of two terms ψ_1 and ψ_2, $\psi_1 \sqcup \psi_2$, is the most general term subsumed by both and the dual notion of *antiunification* of two terms ψ_1 and ψ_2, $\psi_1 \sqcap \psi_2$, is the most specific term that subsumes both. There might be more than one antiunification and more than one unification for two given terms. Also, although an antiunification always exist, two terms might not unify. After this summary, we are able to define the *amalgam* of two terms.

2.1 Amalgam-based Case Reuse

An *amalgam* of two terms is a new term that contains *parts from these two terms*. For instance, an amalgam of 'a red French sedan' and 'a blue German

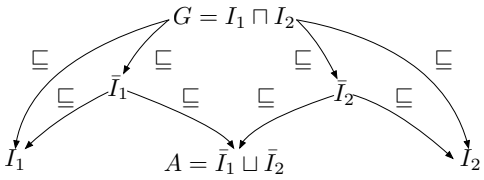

Fig. 1. A diagram of an amalgam A from inputs I_1 and I_2 where $A = \bar{I}_1 \sqcup \bar{I}_2$.

minivan' is 'a red German sedan'; clearly there are always multiple possibilities for amalgams, like 'a blue French minivan'.

In this paper we define an amalgam in a feature term language $\mathcal{L}$ as:

Definition 1 (Amalgam). *A term $A \in \mathcal{L}$ is an amalgam of two inputs I_1 and I_2, with anti-unification $G = I_1 \sqcap I_2$, if there exist two generalizations $\bar{I}_1$ and $\bar{I}_2$ such that (1) $G \sqsubseteq \bar{I}_1 \sqsubseteq I_1$, (2) $G \sqsubseteq \bar{I}_2 \sqsubseteq I_2$, and (3) $A = \bar{I}_1 \sqcup \bar{I}_2$*

When $\bar{I}_1$ and $\bar{I}_2$ have no common specialization then trivially $A = \top$, since their only unifier is "none". For our purpose we will be only interested in non-trivial amalgams (those different from $\top$) of the input pair, which we call their *amalgam space*. This definition is illustrated in Fig. 1, where the anti-unification of the inputs is indicated as G, and the amalgam A is the unification of two concrete generalizations $\bar{I}_1$ and $\bar{I}_2$ of the inputs; for short we call $\bar{I}_1$ and $\bar{I}_2$ the *transfers* of amalgam A. Usually we are interested only on maximal amalgams of two input terms, i.e., those amalgams that contain maximal parts of their inputs that can be unified. Formally, an amalgam A of inputs I_1 and I_2 is maximal if there is no other non-trivial amalgam A' of inputs I_1 and I_2 such that $A \sqsubset A'$.

In our system, amalgamation is used in the Reuse process to create a new story, by combining a story from the case base with a target specifying desired aspects of the story to be generated.

2.2 Argumentation

Computational argumentation, in the abstract framework, consists of a set of nodes (called arguments, intuitively understood as formulas) and an attack relation among pairs of nodes. An abstract argumentation framework $AF = \langle Q, R \rangle$ is composed by a finite set of arguments Q and an attack relation R among the arguments [4]. For instance, an attack relation written $\alpha \twoheadrightarrow \beta$ means that argument α is attacking argument β. In our previous work [14] on learning by communication we integrated inductive concept learning with a concrete argumentation model in the A-MAIL framework (where A-MAIL stands for argumentation-based multiagent inductive learning). In particular, a case e in the case base could serve to attack an argument as a counter-example $e \twoheadrightarrow \beta$. Here A-MAIL is a concrete argumentation framework, not abstract like Dung's, and one main difference is that while Dung's assume a finite, known set of arguments we assume an open-ended set of arguments. As we shall see, using arguments to revise cases during

the CBR cycle is essentially an open-ended process, since more often than not the knowledge (here in the form of arguments) used to revise cases is external to the CBR system.

Another difference is that during the Revise process, the case is assumed to be a concrete, instantiated formula — while in the A-MAIL framework examples and counterexamples were instantiated, and the other arguments were assumed to be general formulas. The usual definition of an attack $\alpha \twoheadrightarrow \beta$ is that α concludes the opposite of β and $\beta \sqsubset \alpha$ (α is a specialization of β). Section 4 introduces the role of arguments in the Revise process of the CBR cycle.

3 Automatic Story Generation

Compared with the established narrative forms such as prose fiction computer-generated stories are still in their early stage. Despite the recent progress in the area, these stories are still fairly rudimental in terms of both the depth of meanings and the range of their varieties.

Automatic story generation is an interdisciplinary topic focusing on devising models for algorithmically producing narrative content and/or discourse. Story generation is an important area for interactive digital entertainment and cultural production. Built on the age-old tradition of storytelling, algorithmically generated stories can be used in a wide variety of domains such as computer games, training and education. In addition, research in story generation may shed light into the broader phenomena of computational creativity [6].

Different techniques have been studied in story generation, the most common of which is automated planning. Salient examples of planning-based story generation systems include Tale-Spin [11], Universe [9] and Fabulist [16]. By contrast, computational analogy algorithms have not been sufficiently explored in the domain of story generation. An alternative approach is that of using case-based or analogy-based approaches. Examples of this alternative approach are MISTREL [19], MEXICA [15] or the work of Gervás et al. [7], which used case-based approaches, or SAM [21], which uses computational analogy.

Specifically, in this paper we focus on a case-based approach, and address the following problem: given a partially specified story (target), and a collection of fully specified stories (case-base), how can we generate a new story by reusing one of the cases in the case base (source)? This is an important problem in story-generation since it would allow for a significant amount of authorial control over the output of the story generator (controlling the target story), while providing a fully automated way to suggest completed stories based on the target.

4 Argument-based Revision

The Revision process in the CBR cycle introduces knowledge that is external to the CBR system to evaluate and/or improve the outcome of the Retrieve and Reuse processes. The situation is similar to supervised Machine Learning where

an external source (called "oracle") gives new information to the learning system on its output; this feedback is used by the learning system to perform credit and blame assignment on its learnt structures, modify them accordingly, and increase performance over time (i.e. learn from interacting with the supervisor). Now, different forms of interacting with the oracle define different modalities of learning. The most common modality in supervised learning is when, in classification tasks, a system predicts as solution a class for an instance, and the oracle either accepts it as correct or, if not, provides the correct class for that instance. This modality is common in CBR systems in classification tasks, where the oracle "revises" the predicted class when the system is wrong and provides the correct class (thus, the revised case is formed and can be retained in the case base with the correct solution). However, oracles can provide different information: e.g. an oracle can provide a yes/no feedback to the system given an instance, but does not provide the correct answer. In semi-supervised learning approaches, such as reinforcement learning, the oracle provides a numerical value that estimates how good is the solution provided by the system.

For CBR systems in more complex tasks than classification, the Revise phase usually assumes an external oracle (that might be a human expert or a domain model) that can provide a revised solution that is correct, so the system can learn. Other approaches, like "critics" in the CHEF system, are able to detect failures on a recipe and apply repair strategies (e.g. add or remove steps in the recipe [8]). This approach is based on analyzing the failure of the plan being executed (in the real world or a simulated world).

The approach we take is to consider the interaction between system and oracle as a restricted form of dialogue, in which the system provides a tentative solution and the oracle's feedback is an argument attacking the parts of the solution that, according to that oracle, are wrong or unsatisfactory. This is particularly interesting in creative domains, such as storytelling, in which what is a "wrong" output (as in classification) or a "failure" (as in executing a plan), is rather difficult to determine. In the long run, our goal would be to have a dialogue between system and oracle on the features that are positive or negative in a particular story being generated. For the purposes of this paper, we focus on the more simple scenario where (1) the CBR system presents a solution (a story), (2) the oracle's feedback is one or several arguments attacking specific aspects or parts of the story, and (3) the system incorporates arguments (as well as the ones provided by the oracle in previous cycles) and generates a new solution (story) that is coherent with most of the previous oracle's argument. We will presently define the notions of "argument" and generation of new stories coherent with a set of arguments.

4.1 Arguments and attacks

Computational argumentation usually defines an attack relation $\alpha \twoheadrightarrow \beta$ between two logical statements α and β. However, our situation is slightly different, in that we have a story represented as a feature term ψ that is a large and complex structure, and an argument that will attack not the whole story but a particular

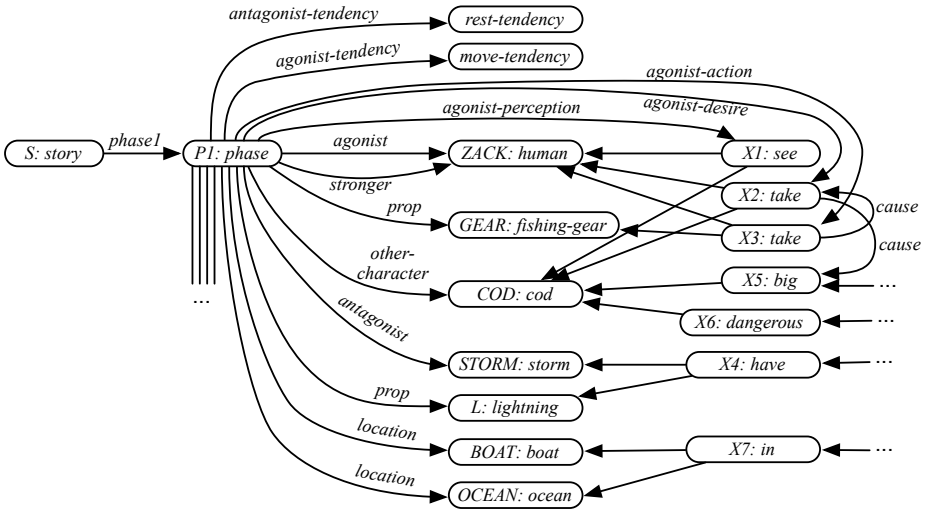

Fig. 2. An example target story used in our experiments (corresponding to the target in S/T3) represented as a feature term. This describes a situation where a human named *Zack*, is in a boat in the middle of the ocean. Zack is taking his fishing gear because he wants to fish a very large cod he has seen. At the same time, there is a storm with lightnings.

part of it (in our formalism, a sub-term of ψ). We will discuss the form of arguments first, and later the attack relation between an argument and an aspect or part of a story.

Definition 2 (Argument). *An argument is a pair (π, α), where π is a term and α is a logical formula over terms with conjunction, disjunction and negation (for terms $\phi, \phi' \in \mathcal{L}$), specifically, α may have one of the following forms: ϕ, $\neg\phi$, $\phi \vee \phi'$, and $\phi \wedge \phi'$.*

Intuitively, an argument (π, α) states that if a story ψ satisfies π (e.g., there is a dragon as antagonist), then the story must also satisfy α (e.g., only a magical weapon can kill the dragon). When the story ψ satisfies π but not α then we say that the argument *attacks* ψ. Moreover, the attacking argument is retained by the system, so in subsequent iterations it would prefer stories that satisfy the argument to those that do not. Therefore, arguments cannot be understood as constraints, but rather as preferences (sometimes called *soft constraints*).

In order to define attacks on stories, represented as feature terms, we need to introduce notation to define subterms of a term. Let $Var(\psi)$ denote the set of variables in term ψ; for instance, the term shown in Fig. 2 has 18 variables, one for each node. Given a variable $X \in Var(\psi)$, the subterm ψ_X is the term with root in variable X, intuitively the subgraph reachable from node X. For instance, in Fig. 2 the variable X_1 has a subterm formed by the root X_1 and the

features that go to *Zack* and *cod*, or if we take the variable P_1 then the subterm is the graph shown in Fig. 2 describing Phase 1 of the story.

Definition 3 (Pattern Satisfaction). *Let π and ψ be terms in $\mathcal{L}$, hereby called* pattern *and* description *respectively. Given a variable $X \in Var(\psi)$, we say a description ψ satisfies a pattern π through X if $\pi \sqsubseteq \psi_X$, we write $\pi \sqsubseteq_X \psi$.*

We can now define an attack of an argument (π, α) against a description ψ; the intuition is that whenever ψ satisfies π then, if ψ does not satisfy the patterns in the formula α, the argument attacks ψ.

Definition 4 (Attack). *An argument (π, α) attacks a description ψ, written $(\pi, \alpha) \twoheadrightarrow \psi$ whenever $\exists X \in Var(\psi)$ such that $\pi \sqsubseteq_X \psi$ and one of the following holds:*

1. *when $\alpha = \phi$ and $\phi \not\sqsubseteq_X \psi$ holds,*
2. *when $\alpha = \neg\phi$ and $\phi \sqsubseteq_X \psi$ holds,*
3. *when $\alpha = \phi \vee \phi'$ and $\phi \not\sqsubseteq_X \psi$ and $\phi' \not\sqsubseteq_X \psi$ hold,*
4. *when $\alpha = \phi \wedge \phi'$ and $\phi \not\sqsubseteq_X \psi$ or $\phi' \not\sqsubseteq_X \psi$ hold.*

Each subsumption relation $A \sqsubseteq_X B$ generates a mapping between $Var(A)$ and $Var(B)$ (as defined in Section 2). The mapping generated when testing subsumption of α must respect the mapping generated for π. Moreover, if π subsumes ψ for more than one mapping, each one of these mappings constitutes a different attack.

For the purposes of this paper we do not use nested logical connectives; our arguments use only simple negation, conjunction and disjunctions. The definition of attack is the converse of satisfaction with respect to the formula α; thus satisfying $\phi \vee \phi'$ means that either one (ϕ or ϕ') is satisfied in ψ, and then there is no attack. Because of this, for $\alpha = \phi \vee \phi'$ accomplishing an attack, it means that neither ϕ nor ϕ' are satisfied in ψ. We will use the notation $|a \twoheadrightarrow \psi|$ to denote the number of variables in $Var(\psi)$ that are attacked by the argument $a = (\pi, \alpha)$.

For the particular case where an argument wants to express that the formula α has to be satisfied regardless of any precondition π, we use the notation $(\bot, \alpha)$.

Finally, notice that a new argument (π, α) is retained by the system, and will be used to generate new stories where the ones that satisfy (are not attacked by) argument (π, α) are preferred. More generally, given a set of known arguments $Args$, the system will generate stories by exploring the space of amalgams and preferring those that satisfy (are not attacked by) more arguments in $Args$.

4.2 Argument-based Revision Algorithm

This section presents a specific Argument-based Revision Algorithm (ARA) that combines the ideas presented above. Specifically, the algorithm we propose performs a greedy search over the amalgam space, starting with the most general amalgam possible (the anti unification of the two input terms I_1 and I_2), and it

Algorithm 1 ARA(I_1, I_2, f, $Args$)

1: $t = 0$, $A_0 = A^* = I_1 \sqcap I_2$, $\bar{I}_1^0 = \bar{I}_2^0 = A_0$
2: **loop**
3: $t = t + 1$
4: $R_1^t = specializations(\bar{I}_1^{t-1})$
5: $R_2^t = specializations(\bar{I}_2^{t-1})$
6: $C = \{I \sqcup \bar{I}_2^{t-1} | I \in R_1^t\} \cup \{\bar{I}_1^{t-1} \sqcup I | I \in R_2^t\}$
7: **if** $C = \emptyset$ **then**
8: **return** A^*
9: **else**
10: $A_t = argmax_{A \in C} evaluation(A, f, Args)$
11: if $evaluation(A_t, f, Args) > evaluation(A^*, f, Args)$ then $A^* = A_t$
12: $\bar{I}_1^t = A_t \sqcap I_1$, $\bar{I}_2^t = A_t \sqcap I_2$
13: **end if**
14: **end loop**

specializes it iteratively, employing arguments to determine which of the possible specializations is the most promising to pursue.

ARA is shown in Algorithm 1, and works as follows. Given two terms I_1 and I_2 (which in our case represent the target case, and the retrieved case), an evaluation function f and a set of arguments $Args$:

1. Step 1 initializes the current amalgam, A_0, the currently best amalgam A^* and the current two transfer terms $\bar{I}_1^0$ and $\bar{I}_2^0$ to be equal to the antiunification of the two input terms (the most general amalgam possible).
2. Then, at each iteration t, first ARA finds the set of possible specializations of the current two transfers (this is done using a *refinement operator* over feature terms [13]).
3. Line 6 computes all the possible next amalgams, resulting from unifying the next specializations with the previous transfer terms.
4. Lines 10 - 11 select the best amalgam, and line 12 updates the transfer terms for the next iteration. The way the best amalgam is determined is where arguments come into play. Each argument a in our framework is assigned a weight w_a. The weight of an argument represents how serious is the issue that this argument tries to prevent[3]. Each amalgam is then assessed as follows:

$$evaluation(A, f, Args) = f(A) - \sum_{a \in Args} w_a \times |a \twoheadrightarrow A|$$

where:
   - f is an evaluation function that provides a basic score for an amalgam. For example, f could encode things like "larger amalgams are preferable" by giving higher scores to amalgams with a larger number of variables.

[3] In the experiments shown later we use a hand-fixed weight equal for all arguments; determining individual weights is discussed in future work.

In our experiments, we used the function: $f(A) = |A| - k \times |Var(A)|$, where $|A|$ is the size of the term A (number of times we need to specialize $\perp$ using the refinement operator to reach A), and captures the size of the story, and $k = 4$ in our experiments. A larger story means that we have been able to transfer more information from the source and target in the amalgam, so a general goal is to maximize $|A|$. The number of variables in A is $|Var(A)|$. When unifying the two transfers, we would like variables of one transfer to be mapped to variables of the other. If this is not the case, the number of variables in the resulting amalgam grows. Thus, minimizing the number of variables in the amalgam has the effect of maximizing the number of variables from the source that are mapped to the target.

- The final term subtracts the weight of each attacked argument multiplied by the number of times the argument attacks some subterm with root X of the story A.

The effect of the ARA algorithm is to find amalgams that strike a balance between maximizing the evaluation function f, and minimizing the number of attacking arguments. The next section describes the generation of stories using amalgams and arguments in an experimental scenario.

5 Experimental Evaluation

In order to evaluate our argument-based revision approach, we prepared an experimental setup that bypasses case retrieval altogether. Thus, we prepared four source/target pairs: S/T1, S/T2, S/T3 and S/T4. In order to compare our results with previous work, we translated the source/target pairs used in our previous work [21] to the feature term formalism used in the approach presented in this paper[4]. Below we provide details on the stories used for evaluation, and then we provide empirical results illustrating the performance of our approach.

5.1 Dataset

As mentioned above, we represent stories using *feature terms* [1]. Specifically, a story is a term composed of a sequence of *phases*, where a phase represents a given instant in a story. Each phase contains a set of characters, locations, props, actions and relations. We separated characters into three groups: the *agonist* (protagonist or main character), the *antagonist* (the main opposing force), and *other characters*. Entities that are not characters are classified into *locations* and *props*. For each of these characters and objects, we specify a collection of properties such as: their relations to other characters, whether they are performing any action, or their desires, likes and possessions.

[4] The specific source/target pairs used for this experiment can be downloaded from https://sites.google.com/site/santiagoontanonvillar/software.

	Source		*Target*			
	phases	*refinements*	*phases*	*refinements*	Surface similarity	Structural similarity
S/T 1	4	203	1	73	low	low
S/T 2	4	203	1	62	low	high
S/T 3	2	90	1	79	high	low
S/T 4	3	134	2	89	high	high

Table 1. Properties of the source-target (S/T) pairs used in our study.

Additionally, the high-level structure of each phase is captured by annotating who is the agonist, the antagonist, and their force relation (inspired in the cognitive linguistics framework of *force dynamics* [18]). This allows us to represent, in a compact way, the high-level structure of a story. Figure 2 shows an example story used in our dataset (the target in S/T3).

As mentioned above, we translated the four source/target pairs used in [21] to feature terms. These four pairs were selected because they represent a variety of scenarios based on how similar the target is to the source. We distinguish two types of similarity between stories: *surface* similarity and *structural* similarity. The former refers to whether two stories contain similar concepts (e.g., both contain a boat and a fish), while the latter refers to whether they have a similar structure (e.g., both refer to a story where the main character overcame a difficulty and succeeded). Surface similarity can be measured by the percentage of keywords shared between two stories, and structural similarity is measured by how much the force dynamics structures representing the stories match.

Table 1 shows some statistics of the stories used in our evaluation. For each story, we show the size in number of phases, and number of refinements (this is the number of times we have to apply the specialization refinement operator to $\perp$ to obtain the given story). This shows that target stories in our dataset tend to be smaller than the source ones (expected, since they are only partially specified). Additionally, we show which source/target pairs are similar in terms of surface and structural similarity.

5.2 Experimental Setup

We evaluated our approach in the following way. First, we run our Reuse approach with an empty set of arguments for each of the four source/target pairs. This gets us a baseline against with which to compare. Then, we iteratively generated arguments to address all the issues we observed in the solution generated for S/T1. This resulted in a total of 15 arguments. Using those arguments, we then report on the performance of the system in generating stories again for all the four source/target pairs. We did not generate arguments for the other source/target pairs purposefully, in order to assess the extent to which arguments generated for one story can be used to improve performance in other stories. To compare results, we report the value of the evaluation function f for the solution found, and the number of attacks that the resulting story receives with the 15

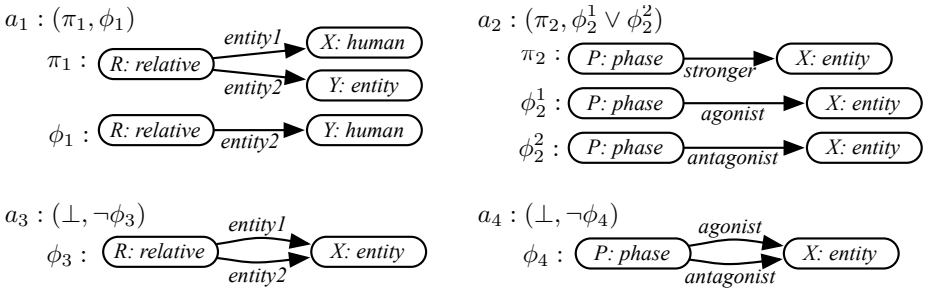

Fig. 3. Four example arguments generated during Revise and used in our experimental evaluation.

arguments we generated. The number of attacks should be seen as a proxy for the number of syntactic or semantic mistakes that the generated stories contain (where a "syntactic mistake" would be something like having a character that is never specified as agonist, antagonist or other-character, and a "semantic mistake" would be something like driving a boat on land). Additionally, we report subjective impressions on the generated stories. Finally, we compare the results obtained with those obtained in our previous work with the same source/target pairs, but with a different algorithm [21].

Figure 3 shows four of the 15 arguments we generated. Specifically, those four arguments capture the following:

- a_1: If a human in the story has a relative, it must also be human.
- a_2: If an entity is marked as "stronger" in a phase (a force dynamics annotations), that entity must be the agonist or the antagonist.
- a_3: An entity cannot be a relative (parent, son, sibling) of itself.
- a_4: An entity cannot be the agonist and the antagonist at the same time.

Notice that those four example arguments are basically capturing things that could be specified using a stricter ontology. However, having them as arguments, allows us to be flexible, and allowing violations in a story in some cases. For example, violating a_1 would allow for fantasy stories where a human has a parent being a magical being; violating a_4 would allow for "split-self" stories, where the main character is both the agonist and the antagonist of the story. Other arguments capture common sense (e.g., a boat cannot be driven on land), or story aesthetics (e.g., the location of a story should not change from phase to phase), but they are not hard constraints, and there are specific stories that do not comply to one of them. The weight w_a of all the arguments was set to 10.

5.3 Results

The left-hand side Table 2 shows the amount of time taken and number of amalgams explored generating stories using amalgams, but without argument-based revision. Automatically quantifying the quality of stories generated using

	Not Using Arguments				*Using Arguments*			
	score	*attacks*	*time*	*explored*	*score*	*attacks*	*time*	*explored*
S/T1	224	9	43s	5435	207	3	83s	5785
S/T2	219	3	232s	4015	210	2	185s	4081
S/T3	157	2	7s	1393	144	2	8s	1463
S/T4	230	6	21s	2056	215	1	13s	1909

Table 2. *Score* achieved (result of using the evaluation function f in the final story), number of *attacks* that the final story receives from the set of 15 arguments we used, *time* taken, and number of amalgams *explored* in the four story pairs used in our experiments.

automatic story generation is an open challenge, and thus, in this paper we report the number of attacks received by the stories using the 15 arguments we generated as a proxy for the number of errors those stories contain (although this is by no means a reflection of their literary quality, it reflects their coherence). We will also list a collection of issues or interesting details of each of the generating stories we observed.

- S/T1: In the resulting story, our system made a series of semantic mistakes: created a story where the father of one of the characters is a butterfly, and when the sister of the main character passed away, the main character threw her into the toilet (since in the source story, a pet fish died and was flushed down the toilet); and a collection of syntactic mistakes: incorrect force dynamics structure, listed the sister of the main character as a prop, and used a location in an incorrect place of the story structure.
- S/T2: The resulting story is almost syntactically correct since the stories share strong structural similarity, however, given that they talk about very disparate things (they share very little surface similarity), the story is rather surreal. In the resulting story, there is a fish inside of a flower in a backyard of the main character (who wants to play a game there); the fish later dies.
- S/T3: In this case, the resulting story is perfectly valid: the main character wants to fish a giant cod, but he ends up not being able to, since the fishing gear breaks while pulling the cod out of the ocean. It only contains a couple of syntactic mistakes.
- S/T4: Source and target here are significant similarity (in one the main character drives a car up a mountain, and in the other he drives a motor-boat in a bay). The resulting story is almost correct, except for a few semantic mistakes: first, in the generated story, a motorboat is driven up a mountain (which clearly cannot be done), and also the bottom of the mountain is "in the main character" (which doesn't make sense), also there appear to be two motorboats instead of one. The rest of the story is coherent: after driving for a while, the main character realizes he did not fill the tank, and needs to turn around.

The right-hand side of Table 2 shows that, obviously, when incorporating the arguments into the search process, the resulting stories receive fewer at-

tacks, since the search process was directed towards parts of the amalgam space containing stories with fewer attacks. For example the output for S/T1 received only 3 attacks (while the same 15 arguments would generate 9 attacks against the story generated without taking them into account). Moreover, even if the 15 arguments were generated with S/T1 in mind, other stories also receive attacks, and thus benefit from these 15 arguments. Also, as the table shows, the time taken and the number of amalgams explored vary from the case when we do not use argument-based revision, but do not significantly increase. Looking closely at the generated stories, we observed the following:

- S/T1: After argument-based revision, all of the semantic mistakes in this story disappeared, since we provided arguments to address each of them. Only two small syntactic problems persisted (a prop and a location were used without being defined in their appropriate place in the feature term). Notice that this illustrates both the strengths and weaknesses of our approach: on the one hand, it is easy to provide arguments that prevent semantic or syntactic mistakes, but on the other hand, given that stories are generated by amalgamating information from source and target, it is not possible to force the resulting story to have something that was not present in the source nor target just by using arguments.
- S/T2: The only aspects that were improved in this story are the syntactic ones, concerning some minor force dynamic structures. The overall story is the same as without arguments.
- S/T3: No changes were observed in this story.
- S/T4: Almost all the syntactic and semantic errors were eliminated in this story: the "bottom of the mountain" is now "in the island", and not "in the main character", and there is a single motorboat instead of two. The only semantic error that remains is the fact that a motorboat cannot be driven up a mountain (since we had no argument to address this, as all arguments were generated just to attack the issues of S/T1).

Comparing the results obtained using argument-based revision with the results obtained using the same four story pairs by the SAM algorithm [21], we observed the following. First, our approach is able to transfer much more information from the source case to the target case. SAM is based on computational analogy (it uses the SME algorithm [5] internally), and only transfers elements of the source that are related in some way (via analogical mapping) to the target. The amalgam-based approach naturally achieves the same result, but can also transfer information that is not mapped directly to the transfer. For example, SAM was barely able to transfer any information at all for S/T2, whereas our approach generates a full story consisting of four phases. Another example is S/T4 where SAM generated a story that did not have the semantic mistake of driving a motorboat up a mountain (since the mountain was not transferred to the final story), but had a different semantic mistake: driving a motorboat *after* it had ran out of fuel. Additionally, being able to use arguments to guide the search process provides our new approach a natural way to guide the generation process toward regions of the amalgam space that contain better stories.

6 Related Work

We already discussed the use of critics in CBR planning systems [8] and how it relates to our approach. A related topic is that of critiquing-based recommenders. The main goal in recommender systems is to acquire, via user feedback, a better model of the user preferences: "Critiquing systems help users incrementally build their preference models and refine them as they see more options" [2]. Probably user-initiated critiquing is the more similar to our approach, where a user is presented with product features that can be selected as candidates to be changed. Our approach is different, allowing richer feedback using arguments. These arguments, in the form of pairs (condition, soft-constraint), are acquired by the system to improve on its own task, in this domain generating stories, not for user personalization. Moreover, the arguments are integrated as a driving force into the search process of generating amalgams of stories during Reuse.

7 Discussion and Future Work

This paper has presented an approach to case revision based on arguments. The main idea is to generate a collection of arguments that *attack* specific aspects of a given solution that we want to prevent. These arguments are kept by the system to prevent the same aspects from appearing in future solutions (albeit as *soft constraints* only). The approach was incorporated into a search-based case reuse framework and evaluated in a story generation task.

Our results indicate that by generating a small collection of arguments, our approach was able to generate stories of higher quality, and that the same arguments generated to attack a specific story were successfully used to increase the quality of a separate set of stories.

Future work includes allowing the system to defend itself against attacking arguments, by generating counter-arguments based on stories in the case base, or even arguments that support a specific aspect of a story, rather than attack it, moving closer to a full-fledged argumentation model, such as in [14]. However, we'd need a larger case base of stories for achieving a richer dialogue. Moreover, providing some support in deciding the weights w_a used for strengthening the arguments remains as future work. Eventually, we would like to model the notion of an *audience* to which to generated story is generated. Value-based argumentation offers this possibility, by associating arguments to values and modeling an audience as a partially ordered set of values. The order among values may help in determining the weights w_a for their associated arguments.

Acknowledgements. This research was partially supported by projects ColInvent (FET-Open grant 611553) and NASAID (CSIC Intramural 201550E022).

References

[1] Carpenter, B.: The Logic of Typed Feature Structures, Cambridge Tracts in Theoretical Computer Science, vol. 32. Cambridge University Press (1992)

[2] Chen, L., Pu, P.: Critiquing-based recommenders: survey and emerging trends. User Modeling and User-Adapted Interaction 22(1-2), 125–150 (2012)
[3] Cojan, J., Lieber, J.: Belief revision-based case-based reasoning. In: Proceedings of the ECAI-2012 Workshop SAMAI: Similarity and Analogy-based Methods in AI. pp. 33–39 (2012)
[4] Dung, P.M.: On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. Artificial Intelligence 77(2), 321–357 (1995)
[5] Falkenhainer, B., Forbus, K.D., Gentner, D.: The structure-mapping engine: Algorithm and examples. Artificial intelligence 41(1), 1–63 (1989)
[6] Gervás, P.: Computational approaches to storytelling and creativity. AI Magazine 30(3), 49–62 (2009)
[7] Gervás, P., Díaz-Agudo, B., Peinado, F., Hervás, R.: Story plot generation based on CBR. Journal of Knowledge-Based Systems 18(4-5), 235–242 (2005)
[8] Hammond, K.: Explaining and repairing plans that fail. Artificial Intelligence 45, 173–228 (1990)
[9] Lebowitz, M.: Creating characters in a story-telling universe. Poetics 13, 171–194 (1984)
[10] López de Mántaras, R., McSherry, D., Bridge, D., Leake, D., Smyth, B., Craw, S., Faltings, B., Maher, M.L., Cox, M.T., Forbus, K., Keane, M.T., Aamodt, A., Watson, I.D.: Retrieval, reuse, revision and retention in case-based reasoning. The Knowledge Engineering Review 20(03), 215–240 (2005)
[11] Meehan, J.: The Metanovel: Writing Stories by Computer. Ph.D. thesis, Yale University (1976)
[12] Ontañón, S., Plaza, E.: Amalgams: A formal approach for combining multiple case solutions. In: Case-Based Reasoning. Research and Development, pp. 257–271. Springer (2010)
[13] Ontañón, S., Plaza, E.: Similarity measures over refinement graphs. Machine Learning 87(1), 57–92 (2012)
[14] Ontañón, S., Plaza, E.: Coordinated inductive learning using argumentation-based communication. Autonomous Agents and Multi-Agent Systems 29(2), 266–304 (2015), http://dx.doi.org/10.1007/s10458-014-9256-2
[15] Pérez y Pérez, R., Sharples, M.: Mexica: A computer model of a cognitive account of creative writing. Journal of Experimental and Theoretical Artificial Intelligence 13(2), 119–139 (2001)
[16] Riedl, M.: Narrative Generation: Balancing Plot and Character. Ph.D. thesis, North Carolina State University (2004)
[17] Smyth, B., Keane, M.T.: Adaptation-guided retrieval: questioning the similarity assumption in reasoning. Artificial Intelligence 102(2), 249–293 (1998)
[18] Talmy, L.: Force dynamics in language and cognition. Cognitive Science 12(1), 49–100 (1988)
[19] Turner, S.R.: A model of creativity. In: The Creative Process: A Computer Model of Storytelling and Creativity. Lawrence Erlbaum Associates, Hillsdale, NJ (1994)
[20] Wilke, W., Bergmann, R.: Techniques and knowledge used for adaptation during case-based problem solving. In: Tasks and Methods in Applied Artificial Intelligence, pp. 497–506. Springer (1998)
[21] Zhu, J., Ontañón, S.: Shall I compare thee to another story? — An empirical study of analogy-based story generation. IEEE Trans. Computational Intelligence and AI in Games 6(2), 216–227 (2014)