Amalgam-based Reuse for Multiagent Case-based Reasoning

Sergio Manzano[1 2], Santiago Ontañón[1], and Enric Plaza[1]

[1] IIIA-CSIC, Artificial Intelligence Research Institute (Spanish Scientific Research Council), 08193 Bellaterra, Catalonia (Spain)
[2] Universitat Autonòma Barcelona, Bellaterra, 08193 Catalonia (Spain)
{sergio,santi,enric}@iiia.csic.es

Abstract. Different agents in a multiagent system might have different solution quality or preference criteria. Therefore, when solving problems collaboratively using CBR, case reuse must take this into account. In this paper we propose ABARC, a model for multiagent case reuse, which divides case reuse in two stages: *individual reuse*, where agents generate full solutions internally, and *multiagent reuse*, where agents engage in a deliberation process in order to reach an agreement on a final solution. Specifically, ABARC is based on the idea of *amalgam*, which is a way to generate solutions by combining multiple solutions into one. We illustrate ABARC in the domain of interior room design.

1 Introduction

The multiplicity of experience sources requires to investigate new approaches to reuse and adapt them in the cycle of case-based problem-solving. This paper focuses on the collaboration of multiple CBR systems dealing with complex structure solutions. Each individual CBR system may have different biases on the solutions it prefers — bias may be due to history, preferences, priorities, etc. Therefore, each individual may come to a different solution for the same problem, and might consider a given solution as more or less satisfactory. A collective agreed-upon complex solution cannot be achieved by simple aggregation methods like voting, and requires a richer deliberation process to reach a collective solution. This deliberation process is in fact a joint reuse process that combines both the experiences and the bias of these individuals.

As a motivating example, consider two people who need to agree on the configuration of an office for having good working conditions. Naturally, each person will have their own preferences with respect to what constitutes "good working conditions," or with respect to which pieces of furniture are more desirable, etc. It is clear that proposing room configurations and voting over them is not a good approach, since the number of possible room configurations might be very high. Moreover, notice that biases such as preferences are hard to express explicitly, and thus cannot be fully communicated. For that reason, a centralized approach where each person states her preferences and then only one of them tries to find a room configuration is not viable either. In this paper we

will study how can these kind of problems be addressed by presenting ABARC, a model of multiagent case reuse, which allows two agents to collaboratively reach an agreement over a structured solution.

Previous approaches to distributed CBR have mainly focused on distributed retrieval [11, 13, 9, 8], while the few approaches dealing with distributed reuse, do so in classification domains [12] by means of voting approaches. The main contribution of this paper is an approach for distributed reuse in domains with complex structured solutions.

There are two key ideas behind the ABARC (Amalgam-based Agreement for Multiagent Reuse of Cases) approach for multiagent CBR. The first is that reuse can be divided in two different stages: *individual reuse* and *multiagent reuse*. During individual reuse, each agent generates a full solution individually according to its own biases or preferences. During multiagent reuse, agents reach an agreement on a solution that is satisfactory for both agents.

The second idea is that the solutions proposed by the other agent can be used as an indication of its preferences. By combining elements from the solutions proposed by the two agents, agents could generate solutions which are satisfactory to both of them. A way to compute such combinations is through *amalgams* [10], a formal operation that generates new solutions by combining as many elements from two given solutions as possible.

The remainder of this paper is organized as follows. First, in Section 2 we present some background on generalization spaces, which is the knowledge representation formalism used in this paper. Section 3 formally introduces the idea of an amalgam. Then, ABARC is presented in Section 4. We analyze the behavior of ABARC in the task of room design in Section 5. The paper closes with related work and conclusions.

2 Background

In this paper we will make the assumption that solutions in cases are terms in some language $\mathcal{L}$, and that there exists a *subsumption* relation among terms.

We say that a term ψ_1 subsumes another term ψ_2 ($\psi_1 \sqsubseteq \psi_2$) when ψ_1 is more general (or equal) than ψ_2[3]. The subsumption relation induces a partial order in the terms in a language $\mathcal{L}$, thus, the pair $\langle \mathcal{L}, \sqsubseteq \rangle$ is a *poset* (partially ordered set) for a given set of terms $\mathcal{L}$; additionally, we assume that $\mathcal{L}$ contains the infimum element $\bot$ (or "any"), and the supremum element $\top$ (or "none") with respect to the subsumption order. In the rest of this paper we will call a pair $\langle \mathcal{L}, \sqsubseteq \rangle$ a *generalization space*.

Given the subsumption relation, for any two terms ψ_1 and ψ_2 we can define their *unification*, $(\psi_1 \sqcup \psi_2)$, which is the *most general specialization* of two given terms:

$$\psi_1 \sqcup \psi_2 = \psi : (\psi_1 \sqsubseteq \psi \ \wedge \ \psi_2 \sqsubseteq \psi) \wedge (\nexists \psi' \sqsubset \psi : \ \psi_1 \sqsubseteq \psi' \ \wedge \ \psi_2 \sqsubseteq \psi')$$

[3] In machine learning terms, $A \sqsubseteq B$ means that A is more general than B, while in description logics it has the opposite meaning, since it is seen as "set inclusion" of their interpretations.

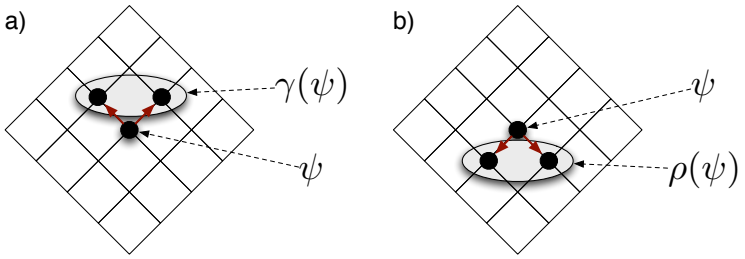

Fig. 1. A generalization refinement operator γ, and a specialization operator ρ.

That is to say, the unifier's content is the addition of the content of the two original terms. However, not every pair of terms may be unified: if two terms have contradictory information then they have no unifier $\psi_1 \sqcup \psi_2 = \top$ —which is equivalent to say that their unifier is "none".

The dual operation to unification is that of *anti-unification*, that is defined as the *least general generalization* of two terms, representing the most specific term that subsumes both. If two terms have nothing in common, then $\psi_1 \sqcap \psi_2 = \bot$. Thus, anti-unification encapsulates in a single description *all* that is shared by two given terms, and is defined as follows:

$$\psi_1 \sqcap \psi_2 = \psi : (\psi \sqsubseteq \psi_1 \ \wedge \ \psi \sqsubseteq \psi_2) \wedge (\nexists \psi' \sqsupset \psi : \ \psi' \sqsubseteq \psi_1 \ \wedge \ \psi' \sqsubseteq \psi_2)$$

Notice that both anti-unification and unification might not be unique. Let us now summarize the basic notions of *refinement operator* over partially ordered sets and introduce the concepts relevant for this paper (see [6] for a more in-depth analysis of refinement operators). Refinement operators are defined as follows:

Definition 1. *A* downward refinement operator ρ *over a partially-ordered set* $(\mathcal{L}, \sqsubseteq)$ *is a function such that* $\forall \psi \in \mathcal{L} | \rho(\psi) \subseteq \{\psi' \in \mathcal{L} | \psi \sqsubseteq \psi'\}$.

Definition 2. *An* upward refinement operator γ *over a partially-ordered set* $(\mathcal{L}, \sqsubseteq)$ *is a function such that* $\forall \psi \in \mathcal{L} | \rho(\psi) \subseteq \{\psi' \in \mathcal{L} | \psi' \sqsubseteq \psi\}$.

In other words, upward refinement operators generate elements of $\mathcal{L}$ which are more general, whereas downward refinement operators generate elements of $\mathcal{L}$ which are more specific, as illustrated by Figure 1. Typically, the symbol γ is used to symbolize upward refinement operators, and ρ to symbolize either a downward refinement operator, or a refinement operator in general.

Refinement operators can be used to navigate the space of terms using search strategies, and are widely used in Inductive Logic Programming [7]. For instance, if we have a term representing "a German minivan", a generalization refinement operator would return generalizations like "a European minivan", or "a German vehicle". If we apply the generalization operator again to "a European minivan", we can get terms like "a minivan", or "a European vehicle". A specialization refinement operator would perform the opposite task, and given a term like "a

German minivan", would return more specific terms like "a Mercedes minivan", or "a red German minivan".

3 Amalgams

The notion of *amalgam* can be conceived of as a generalization of the notion of unification over terms. The unification of two terms (or descriptions) is a new term, the unifier, which contains all the information in these two terms. Thus, if a term ϕ is a unifier of two other terms ($\phi = \psi_a \sqcup \psi_b$), then all that is true for one of these terms is also true for ϕ. For instance, if ψ_a describes "a red vehicle" and ψ_b describes "a German minivan" then their unification ϕ is the description "a red German minivan." Two terms are not unifiable when they possess contradictory information; for instance "a red French vehicle" is not unifiable with "a red German minivan" since being French and German at the same time is not possible for vehicles. The strict definition of unification means that any two descriptions with only one item with contradictory information cannot be unified. Now, imagine a scenario where two such descriptions have a large part of complementary information, which a CBR system would be interested in reusing: unification is not useful.

An *amalgam* of two terms (or descriptions) is a new term that contains parts from these two terms. For instance, an amalgam of "a red French vehicle" and "a German minivan" is "a red German minivan"; clearly there are always multiple possibilities for amalgams, since "a red French minivan" is another example of amalgam. In [10] we formally defined the notion of amalgam, and specifically studied the most specific amalgams of two terms. For the purposes of this paper, the following, more generic definition of amalgam suffices.

Definition 3. (Amalgam) *The set of* amalgams *of two terms ψ_a and ψ_b is the set of terms such that:*

$$\psi_a \curlyvee \psi_b = \{\phi \in \mathcal{L} - \{\top\} | \exists \phi_a, \phi_b \in \mathcal{L} | \phi_a \sqsubseteq \psi_a \ \wedge \ \phi_b \sqsubseteq \psi_b \ \wedge \ \phi = \phi_a \sqcup \phi_b\}$$

Thus, an amalgam of two terms ψ_a and ψ_b is a term that has been formed by unifying two terms ϕ_a and ϕ_b such that $\phi_a \sqsubseteq \psi_a$ and $\phi_b \sqsubseteq \psi_b$ —i.e. an amalgam is a term resulting from combining some of the information in ψ_a with some of the information from ψ_b, as illustrated in Figure 2. Formally, $\psi_a \curlyvee \psi_b$ denotes the set of all possible amalgams; however, whenever it does not lead to confusion, we will use $\psi_a \curlyvee \psi_b$ to denote one specific amalgam of ψ_a and ψ_b.

The terms ϕ_a and ϕ_b are called the *transfer* terms of an amalgam $\psi_a \curlyvee \psi_b$. ϕ_a represents all the information from ψ_a which is *transferred* to the amalgam, and ϕ_b is all the information from ψ_b which is transferred into the amalgam. This idea of transfer is akin to the idea of *transferring* knowledge from the source domain to the target domain in computational analogy [3].

Intuitively, an amalgam is *complete* when all which can be transferred from both terms into the amalgam has been transferred, i.e. if we wanted to transfer more information, ϕ_a and ϕ_b would not have a unifier.

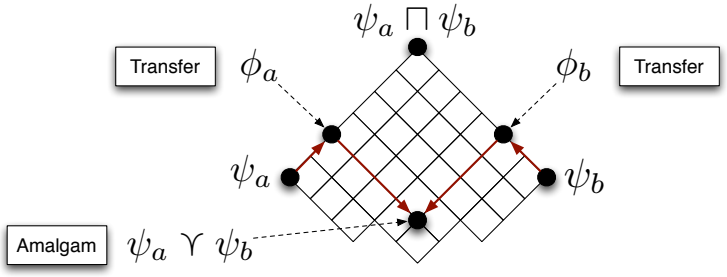

Fig. 2. Illustration of the idea of amalgam between two terms ψ_a and ψ_b.

Definition 4. (Complete Amalgam) *An amalgam $\phi = \psi_a \curlyvee \psi_b$ with transfers ϕ_a and ϕ_b is* complete *when*

$$\forall \phi'_a, \phi'_b | \phi_a \sqsubset \phi'_a \sqsubset \psi_a \wedge \phi_b \sqsubset \phi'_b \sqsubset \psi_b \Rightarrow \phi'_a \sqcup \phi'_b = \top$$

that is to say, there are no terms ϕ'_a , ϕ'_b such that $\phi_a \sqsubset \phi'_a \sqsubset \psi_a$ and $\phi_b \sqsubset \phi'_b \sqsubset \psi_b$ which have a unifier.

Finally, for the purposes of case reuse,we introduce the notion of asymmetric amalgam, where one term is fixed while only the other term is generalized in order to compute an amalgam.

Definition 5. (Asymmetric Amalgam) *The* asymmetric amalgams $\psi_s \overrightarrow{\curlyvee} \psi_t$ *of two terms ψ_s (called* source*) and ψ_t (called* target*) is the set of terms such that:*

$$\psi_s \overrightarrow{\curlyvee} \psi_t = \{\phi \in \mathcal{L} - \{\top\} | \exists \phi_s \in \mathcal{L} | \phi_s \sqsubseteq \psi_s \ \wedge \ \phi = \phi_s \sqcup \psi_t\}$$

In an asymmetric amalgam, the target term is transferred completely into the amalgam, while the source term is generalized.

4 ABARC

This section presents ABARC (Amalgam-based Agreement for Multiagent Reuse of Cases), a framework to address case reuse in multiagent CBR scenarios. Specifically, the scenario we focus on in this paper is the following.

Two CBR agents A_1 and A_2 need to solve a given problem P; the agents have s shared ontology to represent problems and solutions (e.g. solutions from one agent can be understood by the other), but each agent has its own individual biases with which to solve problems, i.e. given the same problem, each agent might prefer different solutions. In ABARC, biases are modeled as *utility functions*. Each agent A_i has a utility function U_i, which given a solution S to the problem P, returns a utility $U_i(S) \in [0, 1]$. These utility functions are private and we assume cannot be communicated (e.g. it is hard to completely communicate to another person you full preferences for food). Given a new problem, the goal

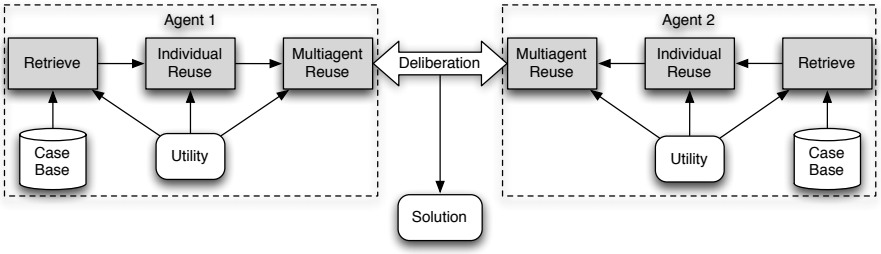

Fig. 3. In the ABARC framework, reuse is divided in two separate processes: individual reuse, and multiagent reuse.

of the agents is to reach an agreement over a solution, which is satisfactory to both agents (i.e. which has a high enough utility for both).

To address the previous scenario, ABARC assumes that problems to be solved have two separate parts:

- *A shared problem specification*: the specification problem that needs to be solved, shared between the agents.
- *An individual problem bias*: the biases or preferences of a specific agent, represented as its own utility function over solutions, which is private to each agent.

Given that no agent knows the utility functions of the other agent, no one can actually propose a solution individually expecting it to have a high utility for the other agent. The main idea behind ABARC is that each agent individually can reuse cases to generate a solution which satisfies both the shared problem specification and its individual problem biases, and then combine the two solutions in order to generate a solution that satisfy at the same time the problem specification and the biases of both agents. Figure 3 illustrates this idea, where the reuse process of two CBR agents is split into two separate processes:

- *Individual reuse*: agents reuse the retrieved cases from the case base and generate a first candidate solution to problem P, which is passed on to the second reuse stage. We will call the solution generated by the individual reuse stage the *initial solution* of each agent.
- *Multiagent reuse*: agents start a deliberation process, where they propose solutions that combine parts of the agents' initial solutions (using amalgams) in order to generate a solution which satisfies as much as possible the utility functions of both agents. This is done in this way, since the initial solutions are the only indication an agent has of the utility function of the other agent.

Basically, the multiagent reuse step explores the space of amalgams between the initial solutions proposed by the agents. Given that this space might be very large, we propose a deliberation protocol that proceeds in a series of rounds

where agents propose solutions, which are then evaluated by the other agent. The process continues until one agent accepts the solution of the other agent.

Assuming that solutions proposed by an agent A_i have high utility for A_i, if an agent A_j wants to generate a solution with expected high utility for A_i, such solution should be similar to the one proposed by A_i. This is the second key idea behind ABARC: initially agents generate solutions that have high utility for themselves, and at each round, the solution proposed by an agent contains more and more elements from the solution proposed by the other agent. This is done by computing amalgams that transfer more and more elements from the solution proposed by the other agent. The more elements are transferred from the other agent's solution, the less elements can be kept from the agents desired solution. Thus, the multiagent reuse process can be seen as a *concession process* where agents propose solutions that lower one's utility while trying to approach the interests of the other agent, until one agent accepts the solution of the other.

The remainder of this section first presents a specific interaction protocol that implements the deliberation process required to reach a final agreement over the solution to the problem P, and then how can the amalgam operation be used to merge solutions generated by both different agents.

4.1 Multiagent Reuse Interaction Protocol

The goal of the multiagent reuse process is to reach an agreement over a solution for the problem P. To reach such agreement, the agents explore the space of possible amalgams between their two individual solutions (which is the subspace of the whole solution space which is expected to have high utility for both). Moreover, it is possible that no solution exists that has maximum utility for both agents. Thus, it is important that agents reach an agreement, and settle for a "good enough" solution. In order to achieve that, ABARC uses a deliberation protocol where the agents are allowed to send three kind of messages:

- $propose(A, S)$: with which an agent A proposes solution S to the other agent.
- $accept(A, S)$: with which an agent A accepts the solution S previously sent by the other agent.
- $withdraw(A)$: with which an agent A withdraws from the protocol before reaching an agreement.

Given two agents, A_1 and A_2 who are trying to reach an agreement on the solution of a given problem P, the ABARC protocol is a token passing protocol, where at each round t, one of the two agents holds a token. Let us call S_1^0 and S_2^0 the solutions found by A_1 and A_2 respectively after performing individual reuse. S_1^0 is a valid solution for P and that has a high utility value for A_1, but might have a low utility value for A_2 (resp. S_2^0).

1. Initially, A_1 sends the message $propose(A_1, S_1^0)$, and A_2 sends the message $propose(A_1, S_2^0)$. Then, a token is given to one agent at random, round $t = 1$ starts, and the protocol moves to step 2.

2. The agent A_i owner of the token does the following:
   (a) Generates a new solution S_i^t. If a new solution cannot be generated (e.g. when the whole space of amalgams has already been explored), then A_i sends the message $withdraw(A_i)$ and the protocol ends without reaching any agreement.
   (b) A_i evaluates the utility of all the solutions sent by the other agent in rounds prior to t, and selects the solution S with maximum utility $U_i(S)$.
   (c) If $U_i(S) \geq U_i(S_i^t)$, then A_i will send the message $accept(A_i, S)$, and the protocol ends. Otherwise, A_i sends the message $propose(A_i, S_i^t)$, the token is given to the next agent, a new round $t+1$ starts, and the protocol moves back to 2.

A key step in the protocol is 2.a, where agents generate new solutions. In ABARC the first solution that an agent A_i proposes, A_i^0 only takes into account its individual utility function. At each round, agents propose a new amalgam, which transfers a bit less from their solution, and a bit more from the other's solution, i.e. they propose solutions with higher expected utility for the other agent. In this way, agents *approach* each other, until they reach an agreement.

Therefore, the key component in the protocol is the mechanism for generating new solutions (new amalgams), as explained in the next section.

4.2 Merging Solutions Through Amalgams

Two key functionality required in ABARC is generating solutions during the multiagent reuse protocol. Let us first explain how can this be implemented using amalgams and refinement operators.

Given a solution S_i generated by an agent A_i, and a solution S_j generated by an agent A_j, if an agent A_i wants to generate a solution S^* which results from merging S_i with S_j, but without losing utility, it can do so in the following way, by using the previously introduced idea of asymmetric amalgam:

1. Let $B = \{S \in \mathcal{L} | S \sqsubseteq S_j \ \wedge \ (S_i \sqcap S_j) \sqsubseteq S\}$ be the set of potential transfers that can be used for computing the asymmetric amalgam $S_j \overrightarrow{\Upsilon} S_i$.
2. Let $\phi_j = \arg\max_{\phi \in B} U_i(\phi \sqcup S_i)$. Notice that $\phi \sqcup S_i$ is an asymmetric amalgam $S_j \overrightarrow{\Upsilon} S_i$, where ϕ is the transfer. Thus, ϕ_j is the transfer which maximizes the utility of the resulting amalgam for A_i.
3. $S^* = \phi_j \sqcup S_i$.

The previous method is generic, and can be used for any utility function. However, in the experiments reported in this paper we have modeled the utility functions in the following way. Each agent has a lists of private goals $G_1 = \{g_1^1, ..., g_n^1\}$, and $G_2 = \{g_1^2, ..., g_m^2\}$ respectively, where each goal g_j^i has an associated weight $w(g_j^i) > 0$, indicating how important it is (the higher the weight, the more important the goal is). A goal g is satisfied by a solution S if $g \sqsubseteq S$, therefore, if we have two solutions, S_1 and S_2, such that $S_1 \sqsubseteq S_2$, and g is satisfied by S_1, then g is also satisfied by S_2.

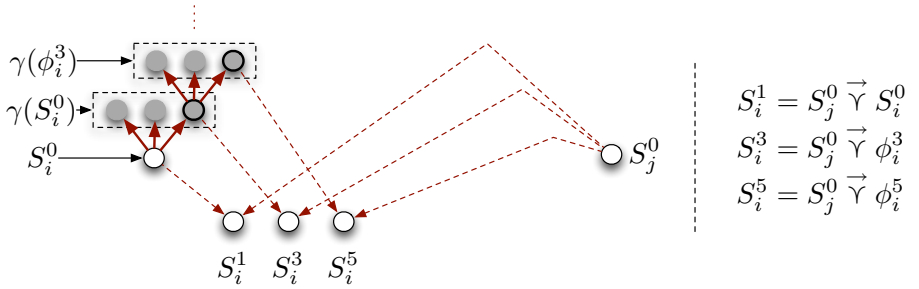

Fig. 4. Solutions during the mutliagent reuse protocol are generated by an agent by iteratively conceding more and more, i.e. by transferring less and less from its own solution.

Therefore, any asymmetric amalgam will satisfy that $S_i \sqsubseteq (S_j \overrightarrow{\Upsilon} S_i)$, and thus will have, at least, the same utility as S_i. Moreover, since it is in the agent's own interest to propose solutions that maximize the other agent's utility as well, the more specific the amalgam, the higher the utility is likely to be for the other agent, and thus, it is desirable to compute complete amalgams.

In addition to merging solutions, agents in ABARC need to generate different amalgams in each round of the protocol, approaching the other agent. The way this is modeled in ABARC is by generating amalgams between S_i^0 and S_j^0 that are not asymmetric, i.e. where not all the information in S_i^0 is transferred to the amalgam. In an asymmetric amalgam $S_j^0 \overrightarrow{\Upsilon} S_i^0$, the transfer from S_i to the amalgam is $\psi_i = S_i$. A way to generate a solution that approaches that of the other agent is by transferring less information from S_i, i.e. having a transfer $\phi_i \sqsubset S_i$.

Moreover, A_i only wants to concede the minimum amount necessary. Thus, given that at a round t, an agent A_i proposed the solution S_i^t, with transfers ϕ_i and ϕ_j form S_i^0 and S_j^0 respectively, and given a generalization refinement operator γ, $\gamma(\phi_i)$ is a set of terms which are more general than ϕ_i. Given any term $\phi_i' \in \gamma(\phi_i)$, if we compute a complete asymmetric amalgam $S_j^0 \overrightarrow{\Upsilon} \phi_i'$, we obtain an amalgam with less transfer from S_i^0 than S_i^t (and thus may accommodate more transfer from the solution of the other agent), i.e. we obtain a solution that approaches that of the other agent.

In ABARC, agents generate new solutions during the protocol in the following way, as illustrated in Figure 4:

1. In the initial round, the agent proposes its initial solution S_i^0.
2. Then, the first time an agent A_i owns the token, it will propose the solution generated by a complete asymmetric amalgam $S_j^0 \overrightarrow{\Upsilon} S_i^0$.
3. The subsequent times an agent A_i owns the token, it will generate a solution by generalizing the last transfer term using a refinement operator:

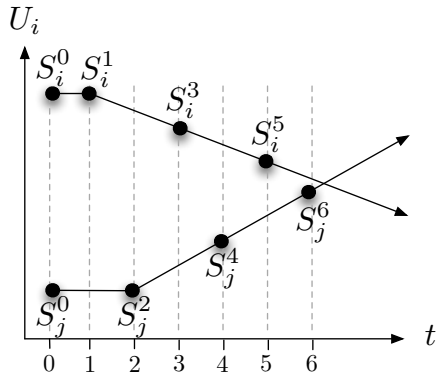

Fig. 5. The utility for one agent A_i of the solutions proposed by both agents over time.

(a) First, compute $\phi_i' = \arg\max_{\phi \in \gamma(\phi_i)} U_i(\phi)$, which is the generalization of the previous transfer ϕ_i with highest utility for A_i.
(b) The next solution to propose will be $S_j^0 \overrightarrow{\Upsilon} \phi_i'$.

As Figure 4 shows, by iteratively generalizing the transfer term, the solutions generated by an agent A_i contain less and less from its original solution, S_i^0, in order to accommodate more and more from the solution proposed by the other agent, S_j^0. The effect is the sequence of solutions proposed by A_i have decreasing utility for A_i, but increasing for A_j, and the solutions proposed by A_j have increasing utility for A_i and decreasing for A_j. As illustrated in Figure 5, the utility values will eventually cross, and the agents will reach an agreement.

5 Collaborative Generation of Room Designs

In order to illustrate ABARC, we have applied it to the domain of interior room design. In this domain the goal is to design the interior of a room (pieces of furniture to use and their layout) according to some given constraints (size and shape of the room), and a set of given goals (e.g. design a proper work environment). Moreover, we have specifically focused on the problem of designing room for two individuals. In this case, when designing the room, the two individuals (represented by agents) have to agree on a design that satisfies both.

In this section we will present an illustration with a particular instance of this problem, in which two agents need to agree on the design of a square office for work space. Agent A_1 has a "geek" personality, and agent A_2 represents an "artist" personality. In this instance, we have represented a room to be composed of 8 spaces: the four corners (which can accommodate small pieces of furniture) and the four big walls (which can accommodate large pieces of furniture). The south-west corner needs to be free, for the door to enter the room. The agents can select among 7 different small pieces of furniture (like dressers, plants, small

Table 1. Goals of two agents in a room design problem.

Agent A_1		Agent A_2	
Goal	*Weight*	*Goal*	*Weight*
tower-computer	8.0	drawing table	8.0
computer desk	7.0	Tansu	7.0
plant pot	6.0	glass-door cabinet	6.0
bookcase	5.0	couch	5.0
couch	4.0	laptop	4.0
armchair	3.0	plant pot	3.0
dresser	2.0	filing cabinet	2.0
cabinet	1.0	corner desk	1.0

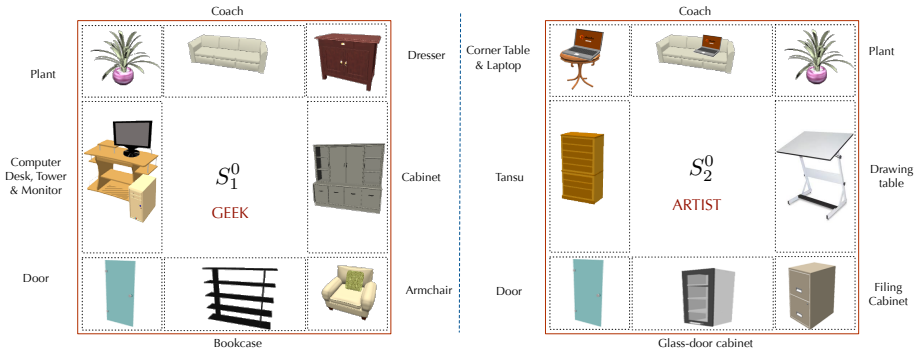

Fig. 6. The two room designs according to the Geek's and Artist's goals.

desks, etc.) and 9 different large pieces of furniture (like couches, desks, etc.). Some pieces of furniture (like computer desks), can also have devices on top (like netbooks, laptops or computers). In total, there are about 1.3×10^9 possible room configurations. For this example, we represented solutions using *feature terms* [1], and we thus use a general feature term generalization refinement operator.

The goals and weights of the goals that compose the utility functions of the two agents are shown in Table 1. Utility is measured in the following way:

$$U_i(S) = \frac{\sum_{g \in G_i | g \sqsubseteq S} w(g)}{\sum_{g \in G_i} w(g)}$$

i.e. as the sum of the weights of the goals that are satisfied by a solution, divided between the total sum of weights of the goals. Thus, if we have two solutions S_1 and S_2 such that $S_1 \sqsubseteq S_2$ then we know that $U(S_2) \geq U(S_1)$.

After solving the problem individually, the agents come up with the solutions shown in Figure 6[4]. If we evaluate the utilities of the solutions for both agents,

[4] Notice that the 2 laptops appearing in S_0^2 means that there are 2 ways to achieve the goal of having a place for a laptop, not having 2 places or 2 laptops.

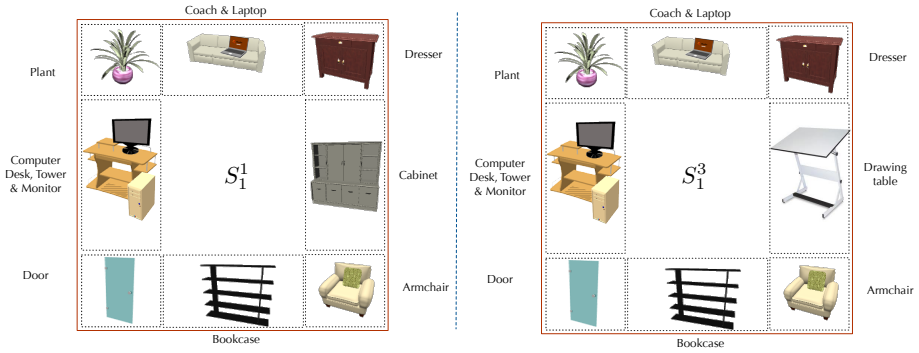

Fig. 7. Solutions S_1^1 and S_1^3 proposed by A_1 in rounds 1 and 3.

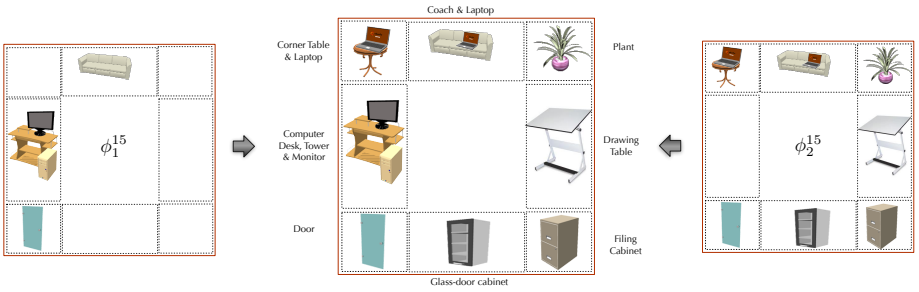

Fig. 8. Final agreed-upon solution, and the transfers from the initial solutions proposed by the agents.

we obtain that: $U_1(S_1^0) = 1.0$, $U_1(S_2^0) = 0.22$, $U_2(S_1^0) = 0.31$, and $U_2(S_2^0) = 1.0$. Notice that the solution proposed by A_2 (the artist) has a non zero utility for A_1 (the geek), since S_2^0 contains elements such as a plant pot, a couch and a glass-door cabinet (which is just a special kind of cabinet), and thus S_2^0 satisfies some of the goals of A_1.

When the ABARC protocol starts (t=1), say agent A_1 owns the token; then A_1 tries to find an asymmetric amalgam $S_2^0 \overrightarrow{\Upsilon} S_1^0$ to propose as a new solution. That is to say, A_1 tries to find a transfer $\phi_2^1 \sqsubseteq S_2^0$ that unifies with S_1^0; the result is the solution S_1^1, shown on the left hand side of Figure 7. This solution keeps all the content of S_1^0 and has incorporated the laptop, which fits on top of the couch. This solution has improved A_2's utility to $U_2(S_1^1) = 0.33$, since one more of its goals (having a laptop) is satisfied. Two rounds later, A_1 needs to generate another solution proposal, and, using a refinement operator, generalizes S_1^0 giving up the *cabinet* in the east wall, to allow any piece of large furniture (ϕ_1^3). Thus, A_1 proceeds to find another asymmetric amalgam $S_2^0 \overrightarrow{\Upsilon} \phi_1^3$, obtaining a solution S_1^3 with the drawing table on the east wall. This solution now has a slightly lower utility for A_1 (0.97), but an increased utility for A_2 (0.56).

After only 15 rounds, they arrive at the solution shown in Figure 8, which has utility 0.72 and 0.81 for A_1 and A_2 respectively. Figure 8 also shows what was transferred from the original solutions by A_1 and A_2. Notice that most of the solution comes from A_2, however, that is fine for A_1, since the part that is transferred from A_2 satisfies many of A_1's goals, such as having a plant-pot or having a cabinet. This shows ABARC can effectively and efficiently help two agents reach an agreed-upon solution even when they have completely different sets of goals, and no agent knows the goals of the other agent.

Moreover, in order to assess how good is the solution found by ABARC, we can compare it with the "best" solution that can be found in the search space. Given that there are two utility functions involved, we may use welfare functions for determining collective utility, such as the *utilitarian* welfare (the solution that maximizes the sum of utilities) or *egalitarian* welfare (the solution that maximizes the minimum utility). Considering a utilitarian welfare, the best solution that could have been found has utility values of 0.875 and 0.75 for A_1 and A_2 respectively. To find that solution, we ran an exhaustive algorithm that explored all the 1.3×10^9 possible solutions. In comparison, agents in ABARC only shared 15 different solutions (and considered 161 solutions internally). Moreover, the solution found by ABARC has a utility about 94% of the optimal solution in utilitarian welfare. This means that ABARC manages to find a very good solution very fast, thanks to the amalgam operation. On the worst case, agents using ABARC would explore the set of all possible amalgams between their initial solutions, which is a subspace of the complete solution space.

6 Related Work

Three areas of work are related to ABARC: reuse in multiagent systems, reusing multiple cases, and work on merging operations.

Concerning reuse in multiagent systems, the first work in multi-CBR systems was presented by Prasad, Lesser and Lander [13]. They focus on a system where a set of individual agents have collected experience on their own, and thus can have a local view of each problem. They present a decentralized "negotiated case retrieval" technique that allows the group of agents to retrieve the appropriate information from each individual case base and then aggregate it in order to answer a query. The aggregation, however, does not correspond to merging solutions as in our framework, but to the fact that different agents might only know some subset of the features of each problem. Similarly, other research in multiagent CBR, like Plaza, Arcos and Martin [11], McGinty and Smyth [9], or Leake and Sooriamurthi [8], focuses on distributing the retrieval stage.

To the best of our knowledge, the only work on distributed reuse is that of Ontañón and Plaza [12], which focuses on classification tasks, and on using voting mechanisms as a distributed reuse procedure. The difference with our work is that we focus here on structured solutions, and thus, voting mechanisms are not appropriate.

Concerning reusing multiple solutions, specially relevant are *compositional adaptation* techniques, which find new solutions from multiple cases and have been analyzed for configuration tasks in [15], where the approach *adaptation-as-configuration* is presented. This approach has two specific operators (compose and decompose) that achieve compositional adaptation in "conceptual hierarchy oriented configuration." *Compose* merges multiple concept instances that have already been configured, while *Decompose* gives the subconcept instances of a given concept instance. These operations work upon *is-a* and *part-of* relations in an object-oriented hierarchy.

Another application, different from configuration, is planning. Systematic Plan Adapter (SPA) [4] is an approach for adapting plans as search in a refinement graph of plans. SPA is systematic and complete and, as local search, is based on adapting a single plan. MPA (Multi-Plan Adapter) [14] is an extension which allows for reusing multiple plans. MPA breaks the different plans into smaller pieces, which are then recombined together. The complexity breaking these plans into smaller pieces, however, is very high and MPA uses only a heuristic. Compared to MPA, our approach avoids the need of this breaking down process while providing a systematic way to combine multiple solutions into a single one.

Concerning merging operations, they have been studied in *belief merging* [5], where the goal is to merge two knowledge bases (beliefs plus integrity constraints) while maintaining consistency. This approach was applied to CBR [2] by viewing case combination in reuse as a belief merging problem. Specifically, each case is viewed as a knowledge base, and the merging is generating a new knowledge base that preserves the relevant integrity constraints.

7 Conclusions

In this paper we have focused on distributed reuse in multiagent CBR systems. Specifically, we have focused on the scenario where two CBR agents with different biases or solution quality criteria need to agree on a single structured solution for a given problem. To address this scenario, we have presented ABARC, which models this problem by dividing the reuse process in two subprocesses: individual reuse, and multiagent reuse. During multiagent reuse, we use the *amalgam* operation and refinement operators in order to propose solutions that consist of different ways to merge the two individual solutions proposed by the agents.

ABARC effectively allows agents to reach agreements over structured solutions by exploring the different ways in which the solutions proposed by each agent can be merged (by using the amalgam operation). By only exploring the different ways to amalgamate the solutions proposed individually, ABARC avoids an expensive search process over the space of possible solutions trying to find a solution that maximizes the utility for both agents.

The work presented in this paper is one step towards our long term goal of enabling the application of CBR to open multiagent systems. Whereas distributed retrieval has received some attention, the topic of reusing multiple cases in multiagent CBR has remained under-explored. Our future work focuses on two

main lines: on the one hand, we would like to continue analyzing the properties of the amalgam operation for multiple case reuse, and on the other hand, we are interested in the general problem of problem-solving in distributed scenarios, where agents need to reach agreements over potentially structured solutions.

Acknowledgements. This research was partially supported by projects Next-CBR (TIN2009-13692-C03-01) and AT CONSOLIDER CSD2007-0022.

References

[1] Carpenter, B.: Typed feature structures: an extension of first-order terms. In: Saraswat, V., Ueda, K. (eds.) Proceedings of the International Symposium on Logic Programming. pp. 187–201. San Diego (1991)
[2] Cojan, J., Lieber, J.: Belief merging-based case combination. In: Case-Based Reasoning Research and Development (ICCBR'09). Lecture Notes in Artificial Intelligence, vol. 5650, pp. 105–119 (2009)
[3] Falkenhainer, B., Forbus, K.D., Gentner, D.: The structure-mapping engine: Algorithm and examples. Artificial Intelligence 41, 1–63 (1989)
[4] Hanks, S., Weld, D.S.: A domain-independent algorithm for plan adaptation. J. Artificial Intelligence Research 2(1), 319–360 (1994)
[5] Konieczny, S., Lang, J., Marquis, P.: Da2 merging operators. Artificial Intelligence 157(1-2), 49–79 (2004)
[6] van der Laag, P.R.J., Nienhuys-Cheng, S.H.: Completeness and properness of refinement operators in inductive logic programming. Journal of Logic Programming 34(3), 201–225 (1998)
[7] Lavrač, N., Džeroski, S.: Inductive Logic Programming. Techniques and Applications. Ellis Horwood (1994)
[8] Leake, D.B., Sooriamurthi, R.: When two case bases are better than one: Exploiting multiple case bases. In: ICCBR. pp. 321–335 (2001)
[9] McGinty, L., smyth, B.: Collaborative case-based reasoning: Applications in personalized route planning. In: ICCBR. pp. 362–376 (2001)
[10] Ontañón, S., Plaza, E.: Amalgams: A formal approach for combining multiple case solutions. In: Case-Based Reasoning. Research and Development, 18th International Conference on Case-Based Reasoning, ICCBR 2010. pp. 257–271 (2010)
[11] Plaza, E., Arcos, J.L., Martín, F.: Cooperative case-based reasoning. In: Weiss, G. (ed.) Distributed Artificial Intelligence Meets Machine Learning. Learning in Multi-Agent Environments, pp. 180–201. No. 1221 in Lecture Notes in Artificial Intelligence, Springer-Verlag (1997)
[12] Plaza, E., Ontañón, S.: Ensemble case-based reasoning: Collaboration policies for multiagent cooperative cbr. In: Watson, I., Yang, Q. (eds.) In Case-Based Reasoning Research and Development: ICCBR-2001. pp. 437–451. No. 2080 in LNAI, Springer-Verlag (2001)
[13] Prassad, M.V.N., Lesser, V.R., Lander, S.: Retrieval and reasoning in distributed case bases. Tech. rep., UMass Computer Science Department (1995)
[14] Ram, A., Francis, A.: Multi-plan retrieval and adaptation in an experience-based agent. In: Leake, D.B. (ed.) Case-Based Reasoning: Experiences, Lessons, and Future Directions. AAAI Press (1996)
[15] Wilke, W., Smyth, B., Cunningham, P.: Using configuration techniques for adaptation. In: Case-Based Reasoning Technology, Lecture Notes in Computer Science, vol. 1400, pp. 139–168. Springer-Verlag (1998)