ACook: Recipe adaptation using ontologies, case-based reasoning systems and knowledge discovery

Sergio Gutiérrez Mota and **Belen Diaz Agudo** [1]

Abstract.

This paper presents ACook, a program that addresses the problem of adapting the ingredients of a cooking recipe using a variety of techniques like ontologies, CBR systems and knowledge discovery. Using this technologies, we developed three different versions of the tool: Onto-ACook that only uses an ingredient ontology, CBR-ACook which is a case-based-reasoning system, and AKD-ACook based on adaptation knowledge discovery. An adaptation example is used to evaluate the results of the three subsystems and the conclusions are presented. Finally we propose some improvements for each version of the program.

1 Introduction

In the last years of artificial intelligence a lot of tools have been developed in order to solve problems which are increasingly complex. This includes the use of ontologies to formalize knowledge in a particular domain and reason about it. The development of case-based reasoning systems provides a model which we can use to solve problems using older solutions stored in a case-base. Finally, knowledge discovery is a field that describes methods to extract knowledge of a database automatically.

This paper describes ACook, a tool that addresses the challenge of adapt a cooking recipe using specific constraints over the ingredients.

This paper presents and compares three different ways to approach the problem of adapting cooking recipes. The first system, Onto-ACook, works using an ontology of ingredients and only takes into account each banned ingredient separately. The second version, CBR-ACook, uses a case-based reasoning system which needs an expert to add the knowledge of the domain into the program. The last implementation, AKD-ACook, extracts automatically ingredient adaptation patterns using a recipe database and uses them to build a consistent solution ,i.e., one completely based on an unique recipe.

Finally, this document compares the three different systems and outlines their strengths and weaknesses in order to suggest improvements for each one.

2 Technologies

In this section, we present the more basic concepts of the technologies used in ACook and how are being used in the program.

2.1 Ontologies

An ontology is a representation of knowledge into a concrete domain. It contains a set of concepts and the relationships between those concepts. It is used to reason about these concepts and extract properties about the entities within the domain. Figure 1 shows the hierarchy of the main ontology used in ACook that links ingredients by their origin.

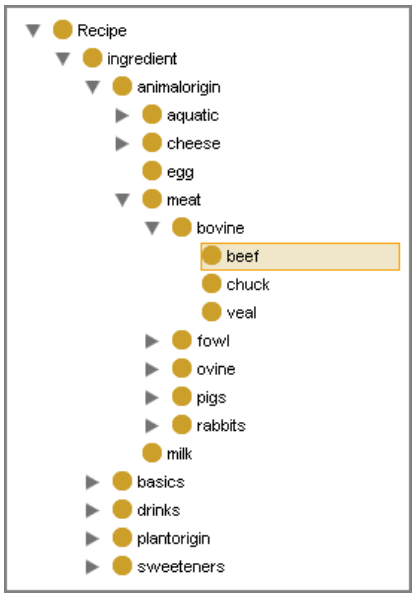

Figure 1: Ingredients ontology used in ACook

2.2 Case-Based Reasoning

Case-Based reasoning (CBR) is a model used to solve problems of artificial intelligence. The main component of the CBR systems are the **cases**, which are a description of a particular problem and its solution. Given a new description of a problem, a CBR system offers a solution based on the similarity to already solved problems stored in the system. The main difference between CBR systems and rules-based systems are that in the first ones it is not needed to formalize the knowledge from an expert, it is just needed to fill the program with descriptions of previous events.

Figure 2 shows an example of a case used in ACook to adapt recipes. Each case consists of the description of the problem (recipe name, used ingredients, preparation and banned ingredients) and its solution. The details of the cases are explained in section 3.2.

2.3 Knowledge Discovery

Knowledge Discovery (KD) is a large field in computation that describes methods to extract knowledge by searching patterns in large

[1] School of Computing. Complutense University of Madrid. Spain, email:belend@sip.ucm.es

Case	
Recipe	*Roasted peaches with brown sugar and sherry*
Ingredients	Butter
	Peaches
	Sherry
	Brown Sugar
Preparation	Cut nectarines in half, remove stones and place, cut side up, on baking sheet. Divide butter among halves, placing little in each cavity. Add 1 teaspoon each Sherry and brown sugar to cavity of each nectarine half. Broil 4 to 6 inches from heat source until topping is bubbly and fruit softened, about 5 minutes.
Banned ingredients	Brown Sugar
Solution	Sugar+

Figure 2: Case example of the CBR system used in ACook

databases. The techniques included in KD depends strongly on the domain of the processed data and the way the programmer wants to show the solutions. The most used branch of knowledge discovery is the one used in databases (KDD) which defines a number of steps to get the knowledge from the data: selection, pre-processing, transformation, data mining and interpretation/evaluation.

ACook uses KD [10] to adapt cooking recipes as it is shown in section 3.3. To understand how it works we need to introduce the basics and some new concepts.

We define a formal context K as a tuple of the form (G, M, r), where G is a set of objects, M is a set of items and r is the relation $G \times M$ meaning that an object is described by an item.

An itemset I is defined as a subset of items of M. Also, the support of I, or support(I) is the number of objects of G having every item of I. Finally, I is called frequent whenever his support is larger than a threshold σ and it is called closed whenever it has no proper superset J with the same support. The idea behind the frequent closed itemsets (FCIs) is that FCIs are the biggest patterns (with more items) used in larges groups of objects.

Figure 3 shows a table with an example of formal context. G is the set of recipes and M is the set of ingredients. Every check in the table means that the recipe uses the ingredient. The only FCI with support 4 is {Onion, Oil}, in other words, there are 4 recipes including onion and oil.

3 Implementation

The following sections presents different versions of ACook and how they use the technologies in the preceding section. In order to make things clear we use the same adaptation query for the three systems: Adapt the recipe called *Greens with Vinaigrette* which uses the ingredients {Lettuce, Vinegar, Pepper, Onion, Oil, Salt, Sugar} and with the following banned ingredients: {Lettuce, Onion}.

Figure 4 shows the application interface, in the first menu the user can choose a recipe from the list and in the second one the user can

	Lettuce	Vinegar	Pepper	Onion	Oil	Salt	Sugar	Garlic	Parmesan	Cherry	Lemon Juice	Chicken
Recipe 1	X	X	X	X	X	X	X					
Recipe 2	X	X		X	X	X		X	X			
Recipe 3	X		X	X	X	X			X	X	X	
Recipe 4			X	X	X		X				X	X

Figure 3: Example of formal context with recipes and ingredients

check the banned ingredients. Each button from the bottom image executes one of the three adaptation subsystems that are described in the following sections.

3.1 Onto-ACook

The first version of ACook uses an ontology of ingredients to build the best adaptation for a recipe. The ontology is an extended version of the one used in JaDaCook [2]. It forms an ingredient hierarchy that contains 221 ingredients arranged in 54 classes by its source and type. The ontology does not take advantage of other features like attributes or restrictions in order to simplify its construction. Figure 1 shows the ontology used in ACook. The algorithm 1 shows how Onto-ACook adapts a recipe using the ontology. For each banned ingredient, the system searches between its closer brothers in the ontology a valid substitution (not already banned or used) and chooses one randomly. When we query the system with the example recipe, *Greens with Vinaigrette*, it informs us that we can make the following replacements:

- Add {Romaine, Asparagus}
- Remove {Lettuce, Onion}

Onto-ACook only gives simple substitutions, where an ingredient can only be replaced by another one and does not take into account more complex possibilities like removing or replacing other non banned ingredients in order to build a better recipe. This is because the substitution system only consider each ingredient separately and does not know anything about the recipe it is used in. In other words, it is unable to handle combinations of ingredients. Furthermore, this version of ACook is very ontology dependent, therefore, the larger and complex the ontology is, the better solutions that Onto-ACook offers. In addition, there is other features like ingredients flavor or quantities that are impossible to capture in ontologies and therefore in Onto-ACook.

3.2 CBR-ACook

The second version of ACook has been fully developed with JColibri [3], a Java framework to create CBR systems. Each case, as shown in figure 2, has the description of a recipe (its name and a list of used ingredients) and a list of banned ingredients. Proposed solutions by CBR-ACook are a list of ingredients that should be added (written with a $+$ symbol) and a list of ingredients that should be removed (written with a $-$ symbol) from the original recipe. It is assumed that banned ingredients are already chosen to be removed from the original recipe so they are not shown in the solution. This new way of expressing solutions gives us more complex adaptations since we can remove ingredients which have not been banned or add more than one ingredient for each substitution.

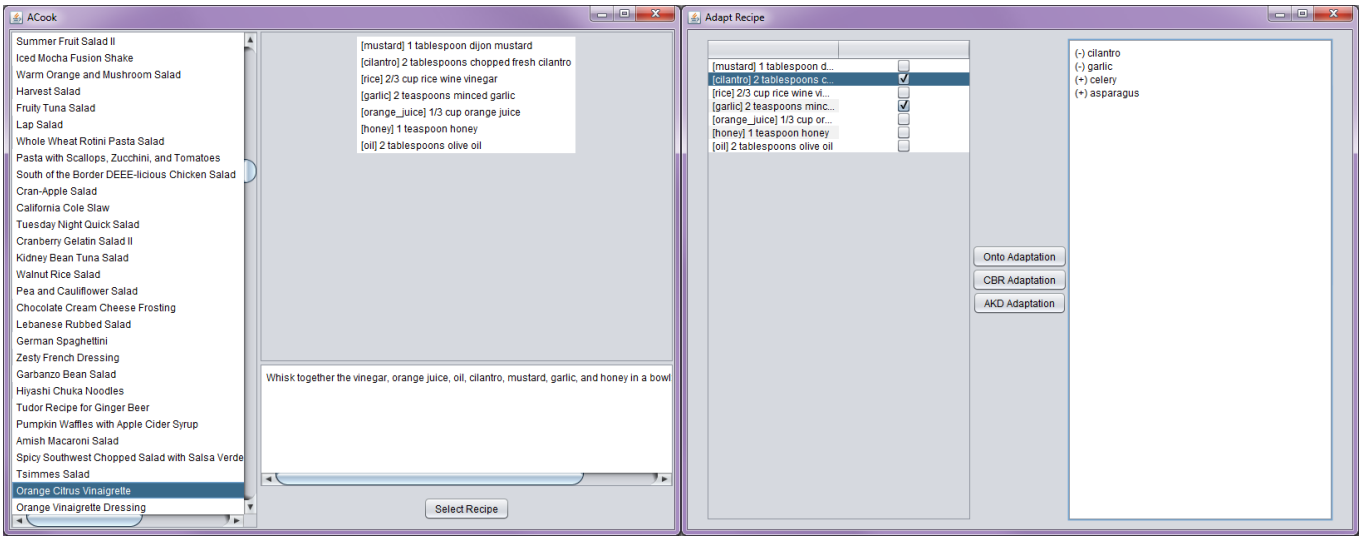

Figure 4: ACook gui

Algorithm 1 Adaptation of Onto-ACook

```
function OntoACook-Adapt(banned_ingredients)
    for all ing ∈ banned_ingredients do
        pad ← ontology.getParent(ing)
        replaced ← false
        while not replaced do
            search for a valid substitution (not already banned or used) in pad parents and updates the replaced one
            if not replaced then
                pad ← ontology.getParent(pad)
            end if
        end while
    end for
end function
```

Algorithm 2 shows how the substitution of CBR-ACook works. One of the keys of CBR systems is the function used to compute the similarity between cases, CBR-ACook does the mean between two values: the similarity between the list of used ingredients and the list of banned ingredients. The similarity between lists of ingredients uses the *deep basic* [6] algorithm, that returns a similarity value of two ingredients according to the distance between them in an ontology. The similarity function is based on the ontology used in Onto-ACook. Algorithm 3 shows a simplified version of the similarity function used to compare lists of ingredients.

Algorithm 2 Adaptation of CBRACook

```
function CBRACook-Adapt(recipe, banned_ingredients)
    query ← recipe + banned_ingredients
    cbr_cycle(query)
    return most_similar_case_solution
end function
```

When we query CBR-ACook with the example recipe, the system suggests the following solution based on previous cases filled by an expert:

- Add {Romaine, Endive, Scallion}
- Remove {Lettuce, Onion}

Algorithm 3 Similarity function of CBRACook

```
function CBRACook-Similarity(query_ingredients, case_ingredients)
    similarity ← 0
    for all query_ing ∈ query_ingredients do
        max_similarity ← 0
        for all case_ing ∈ case_ingredients do
            max_similarity ← max(max_similarity, deep basic(query_ing, case_ing)
        end for
        similarity ← similarity + max_similarity
    end for
    return similarity / number_of_query_ingredients
end function
```

As we can see, this solution is more consistent that the one suggested by the first version of the tool, this is because Onto-ACook only uses each ingredient separately, however, CBR-ACook uses the recipe as a complete entity. Furthermore, the solution contains three new ingredients and only two removed ingredients. This version of the system returns a good adaptation because the expert responsible for filling the cases only worked with salads. It was unfeasible to obtain a complete set of cases for all the recipes and this is one of the main weakness of this type of tools.

3.3 AKD-ACook

The last version of ACook rely on KD techniques applied to adaptation and is completely based on the tool TAAABLE [5]. In order to understand the way AKD-ACook works is necessary to introduce the concept of **recipe variations**. Given two recipes (R_i, R_j), the variations of R_i y R_j are a set of tuples $(\text{ingredient}, \text{modifier})$ called **ingredients variations**. There is only three valid modifiers and its meaning is:

- (*ing*,+) *ing* is used in R_j but not in R_i.
- (*ing*,−) *ing* is used in R_i but not in R_j.
- (*ing*,=) *ing* is used in both R_i and R_j.

Using the two following recipes:

- R_i = {Lettuce, Vinegar, Pepper, Onion, Oil, Salt, Sugar}
- R_j = {Lettuce, Vinegar, Onion, Oil, Salt, Garlic, Parmesan}

variations of (R_i, R_j) are {(Lettuce,=), (Vinegar, =), (Onion, =), (Oil, =), (Salt, =), (Pepper, −), (Sugar, −), (Garlic, +), (Parmesan, +)}. The idea behind this new concept is that the variations of (R_i, R_j) contains the changes that are needed in order to transform R_i into R_j.

AKD-ACook uses the formal context $K = (G, M, r)$ where the set G contains tuples (R, R_i), being R the recipe we want to adapt and each R_i are the other recipes stored in the system. The M set contains every possible ingredient variation and r links G objects with M items following the definition of recipe variations. Figure 5 shows a simple formal context used in AKD-ACook.

As we saw in section 2.3, pattern extraction is the same as FCI extraction. Because this approach is computationally expensive it is common to filter the data in order to handle smaller groups of recipes. In AKD-ACook, the filtering is applied to the recipes of the database which does not satisfy some criteria. Particularly, the recipes used in the FCI extraction process are the ones which satisfy the following tests:

- Does not use any of the banned ingredients.
- Share 50% of ingredients with the original recipe.
- There is no more than 50% of added ingredients compared with the original recipe.

The purpose of this tests is to exclude recipes that are not similar enough to the original one, increasing the amount of potential substitutions that can be considered valid from a cooking point of view.

Using the example recipe and thanks to the filtering process, the number of used recipes in the FCI extraction is reduced from 3385 to 70 and the number of ingredient variations is also reduced from 206 to 51. Once the system has filtered the recipes it creates the formal context and calls the CORON software platform [4] in order to extract FCIs. After this, the program sorts the patterns using five different criteria listed by priority:

1. FCIs with at least one added ingredient.
2. FCIs with more shared ingredients with the original recipe.
3. FCIs with less removed ingredients compared to the original recipe.
4. FCIs with less added ingredients compared to the original recipe.
5. FCIs with higher support.

Finally, the system computes the recipes responsible for the most valued FCI (the ones that use all the ingredients of the pattern). It is needed to use these recipes because the computed FCI is just a pattern of the variations between the original recipe and the others stored in the database, therefore, we cannot guarantee that it forms a consistent recipe. Each recipe involved in the best FCI creates a list of adaptation rules which are displayed to the user. Algorithm 4 shows a simplified version of how the adaptation system of AKD-ACook works.

When the system adapts the example recipe it creates 78 FCIs and chooses the following one as the best valued: {(Vinegar, =), (Pepper, =), (Sugar, =), (Salt, =), (Oil, =), (Celery, +), (Lettuce, −), (Onion, −)} which have a support of 3. The three recipe variations involved in the extraction of the FCI are shown in figure 5. Any of these variations can be used to build the substitution rules and AKD-ACook chooses one of them randomly. In this case it shows the following solution to the user:

Algorithm 4 Adaptation of AKDACook

```
function AKDACOOK-ADAPT(recipe, banned_ingredients, recipes)
    filtered_recipes ← filter(recipes)
    formal_context ← create_formal_context(recipe, banned_ingredients, filtered_recipes)
    fcis ← compute_fcis(formal_context)
    sort(fcis)
    used_recipes ← get_used_recipes(fcis)
    final_recipe ← get_random(used_recipes)
    return build_substitution_rules(final_recipe)
end function
```

- Add {Mustard, Cabbage, Celery}
- Remove {Lettuce, Onion}

Watching the solution obtained with AKD-ACook we can see that the substitution is completely valid because it used real recipes to generate the final adaptation. Furthermore, the solutions of this version of the system are as expressive as the ones computed with CBR-ACook, there can be more than one added ingredient for each banned ingredient. The main advantage of this system compared with the second version is that we did not need an expert responsible for creating the knowledge of the recipes domain since it was obtained automatically. Even the large recipe database was created using a simple program to parse some web pages with all the content. It is also important to note that even using recipes of other types –not only salads– the filtering and sorting steps decrease the possibilities of obtaining misleading results. Finally, it is essential to keep in mind that the adaptation is based on one unique recipe but the choice of this recipe is not made comparing the number of shared ingredients but using the knowledge extracted from the recipe database.

	(Vinegar, =)	(Pepper, =)	(Sugar, =)	(Salt, =)	(Oil, =)	(Mustard, +)	(Dried, +)	(Cabbage, +)	(Celery, +)	(Clove, +)	(Lettuce, -)	(Onion, -)
(R, Recipe 1)	X	X	X	X	X	X		X	X		X	X
(R, Recipe 2)	X	X	X	X	X				X		X	X
(R, Recipe 3)	X	X	X	X	X		X		X	X	X	X

Figure 5: Formal context that contains the recipes used in the best extracted FCI for the example query

4 Conclusions and related work

This paper outlines how the techniques of artificial intelligence are very effective in adaptation problems. With ACook we can see how these techniques can be applied to the same problem and shows the strengths and weaknesses of each one.

It is also sketched that the knowledge about the domain that the system is able to handle is a key point. In Onto-ACook the knowledge was restricted to the ingredients, therefore its recommendations were simpler. CBR-ACook use more knowledge, is able to handle recipes as complete entities, however, needing an expert to complete the knowledge of the system is inviable if we are using a domain as big as the used in this project. The last version of the program, AKD-ACook, is also able to handle recipes as complete entities, therefor its solutions are more complex compared to the first version, but unlike CBR-ACook it is not needed an expert to get all the knowledge, instead, it is extracted automatically.

Each one of the systems can be improved, the ontology can be completed, filling it with more ingredients either adding more attributes to every ingredient. Other works [5] show how the use of several hierarchies is also a viable way to improve the solutions obtained. The CBR system could be improved adding some module to process natural language in order to extract knowledge of the preparation of each recipe. It is also possible to categorize recipes in families so the CBR engine does not have to compare the query with every recipe in the casebase. The last version of the system, AKDACook, can use an extended formal context using the ingredients ontology in order to link ingredients, e.g. ingredients (Lettuce,$-$) and (Romain,$+$) entail the variation (Vegetable,$=$).

REFERENCES

[1] Emmanuelle Gaillard, Jean Lieber, Emmanuel Nauer. Adaptation knowledge discovery for cooking using closed itemset extraction. In *The Eighth International Conference on Concept Lattices and their Applications - CLA 2011*, pages 87-99, Nancy, France, Oct. 2011.

[2] P. Javier Herrera, Pablo Iglesias, David Romero, Ignacio Rubio, Belén Díaz-Agudo. JaDaCook: Java Application Developed and Cooked Over Ontological Knowledge. In Schaaf [Sch08], pages 209-218, 2008.

[3] JColibri: Case-Based Reasoning Framework. *http://gaia.fdi.ucm.es/research/colibri/jcolibri*.

[4] L. Szathmary y A. Napoli. CORON: A Framework for Levelwise Itemset Mining Algorithms. *Supplementary Proc. of The Third International Conference on Formal Concept Analysis (ICFCA '05), Lens, France*, pages 110-113, 2005.

[5] F. Badra et al. Taaable: Text Mining, Ontology Engineering, and Hierarchical Classification for Textual Case-Based Cooking. In *ECCBR Workshops, Workshop of the First Computer Cooking Contest*, pages 219-228, 2008.

[6] J. A. Recio-García, B. Díz-Agudo, P. A. González-Calero, and A. Sánchez-Ruiz-Granados. Ontology based CBR with JColibri. In R. Ellis, T. Allen, and A. Tuson, editors, *Applications and Innovations in Intelligent Systems XIV. Proceedings of AI-2006, the Twenty-sixth SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence*, pages 149-162, Cambridge, United Kingdom, December 2006. Springer.

[7] Ribeiro, R., Batista, F., Pardal, J. P., Mamede, N. J., and Pinto, H. S., Cooking an Ontology. In *Artificial Intelligence: Methodology, Systems, and Applications*, ser. Lecture Notes in Computer Science, no. 4183. Rua Alves Redol, 9: Springer Berlin / Heidelberg, pp. 213-221, Sep. 2006.

[8] F. Badra, A. Cordier, and J. Lieber. Opportunistic Adaptation Knowledge Discovery. In Lorraine McGinty and David C. Wilson, editors, *8th International Conference on Case-Based Reasoning - ICCBR 2009*, volume 5650 of *Lecture Notes in Computer Science*, pages 60-74, Seattle, États-Unis, July 2009.

[9] Aamodt, A. and Plaza, E. Case-Based Reasoning: Foundational Issues, Methodological Variations, and Systems Approaches. In *AI Communications* 7, pp. 39-59, 1994.

[10] U. M. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. From data mining to knowledge discovery in databases. AI Magazine, pages 37–54, 1996.