JADAWeb: A CBR System for Cooking Recipes

Miguel Ballesteros, Raúl Martín and Belén Díaz-Agudo

Department of Software Engineering and Artificial Intelligence, Universidad Complutense de Madrid, Spain.
miballes@fdi.ucm.es, rmb2000@gmail.com, belend@sip.ucm.es

Abstract. JaDaWeb has been developed to participate in the 3rd Computer Cooking Contest 2010. The system has been developed over the system JaDaCook 2.0 that participated in CCC-09. As the main novelties JaDaWeb includes a Web based natural language interface with parsing of the input data, a fuzzy similarity function and Wordnet reasoning to identify and include new ingredients in the ontology. In this paper we present a brief review of the technical characteristics of the system, and some experimental results comparing JADAWeb with Cookiis, the winner system of the CCC-09. The online system is accesible via http://supergaia.fdi.ucm.es:8810/CCCWeb

1 Introduction

In this paper we describe JADAWeb, a CBR system based on JADACook[1, 2] that suggests recipes using as input a set of ingredients to include and to avoid, and some optional dietary and cooking restrictions. JADAWeb retrieves a recipe from the system case base and includes the capability of adapting it by substituting its ingredients by other similar ingredients that appear in the user query. The system uses a fuzzy similarity function to determine if two ingredients can be swapped in the recipe. JADAWeb has been implemented using the jCOLIBRI framework [3] and its facilities to design CBR systems.

JADAWeb has a Web interface built using JSP [4] technology where the input of data is a single character string on natural language in English. We have included an algorithm to detect the negation in a sentence, so the system is able to infer which ingredients the user wants to include and to avoid. The algorithm is based on multilingual dependency parsing[5]. JADAWeb also includes a number of substantial improvements in the decision of which nouns identified in the query are ingredients and which are not. JADACook only checks if the noun appears in the ingredient ontology. However, JADAWeb also searches the noun in Wordnet[6] to determine if the noun is an ingredient or not. If an ingredient is identified in the query and it was not defined in the system ontology, JADAWeb suggests the area of the ontology is better to classify the new ingredient.

After retrieval, JADAWeb suggestions are ordered by similarity and also according with the characteristics of the season. Hotter meals, like a soup, will be presented before in winter, and fresh meals, like salad, will be presented before if the query is in summer. The information that JADAWeb uses to make this

decision is the presence, or not, of some important words that are present in each recipe, like 'hot', 'oven', or 'fresh'.

This paper is organized as follows: Section 2 describes the web-based graphical user interface and the parsing of the input data. In Section 3 we define how the acquisition of knowledge works. Section 4 describes the case-based reasoning process, focusing on the similarity function. Section 5 describes a set of examples tested in the system. We have compared JADAWEb results with the CookIIS system[7] results. Finally, Section 6 shows the conclusions and suggests some lines of future work.

2 Web-Based Graphical User Interface

The web interface allows the user writing a natural language sentence with the recipe requirements (see Figure 1). The following is an example of a query where the user explains what (s)he wants: "I want to eat some fruits, like apple, but I don't like kiwi". The system connects to the dependency text parser to obtain the information contained in this query. It extracts the list of ingredients to include and to avoid in the recipe.

Fig. 1. JADAWeb Web-Based Graphical User Interface

The system infers the lists of ingredients from the query text and shows the result of the parser. The user also can express if he/she wants to choose some dietary practices, or type of cuisine using a form type of interface.

2.1 Textual Parsing of the Input

In this version of JADAWeb the textual input query must be a complete and correct sentence in English. The user query might include the ingredients to include and to avoid in the recipe. We have implemented an algorithm to detect the negation on sentences by using dependency parsing[5]. Dependency parsing

is an important area of research related with artificial intelligence, computational linguistics and machine learning. We have used a dependency parser called Minipar[8] that is limited to English, but it would be possible to use another parsers like Maltparser [9] which allows multilingual dependency parsing. Minipar has an API implemented in C language. Given a plain text sentence, Minipar returns a dependency tree (see Figure 4). Using a wrapper[1] for JAVA we have studied the information given by the parser. Given the tree with the dependency analysis we have implemented an algorithm that detects the attachment of each noun and those who share the same branch of a negation are marked as candidates for unwanted ingredients. All other nouns are marked as wanted ingredients.

The evaluation results show that MINIPAR is able to cover about 79 % of the dependency relationships in the SUSANNE corpus[10] with about 89 % precision[8] for English parsing, so we have a small quantity of errors, but it does not mean that our algorithm committed 10% of errors because is possible that the errors make by Minipar are not important for our purposes.

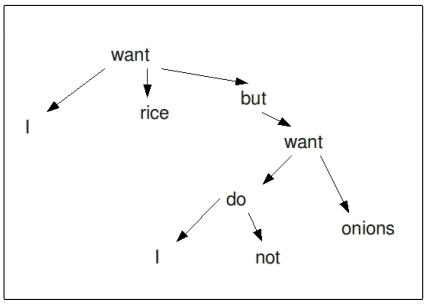

Fig. 2. Dependency tree

Figure 2 shows the information that the Minipar parser returns for the following sentence: 'I want rice but I don't want onions'. Using this information our algorithm builds a structured query that is compared to the recipes in the case base. Although we have found a few labeling errors from the dependency parser, the results obtained by the algorithm are quite reliable in general.

3 Knowledge acquisition

JADAWeb reasons using two main knowledge sources: the recipe base provided by the contest organization, and the ontology that has been built incrementally and collaboratively from the first versions of JADACook. The ontology is conceptualized and formalized in the OWL language[2].

[1] http://nlp.shef.ac.uk/result/software.html
[2] http://www.w3.org/2004/OWL

As a novelty, JADAWeb uses Wordnet[6] to let the user include ingredients that are not previously integrated in the ontology. The textual parsing of the input query searches in Wordnet to decide if a not identified noun in the query is an ingredient or not.

3.1 Ontology

JADAWeb reasons with an ontology that formalizes the cooking domain knowledge. The ontology organizes ingredients in categories. In the first level there are ingredients of animal origin grouped in some subsets: meat, fish, milk, cheese and eggs. There are also, ingredients of plant origin grouped in subsets like cereals, fruits, and vegetables. And finally, there are other specific families of ingredients like sweets, drinks, or basis ingredients like oil or salt.

The ontology in its current state is very extensive and complete, it has 255 classes, 202 of them are ingredients. Although we made little changes to make easier the search and the equality between concept names because Wordnet has different terms than our previous ontology. For example, we changed 'AnimalOrigin' and 'PlantOrigin' for 'animal' and 'plant'.

3.2 Ingredient treatment using Wordnet

Despite the richness of the ontology and the wide range of definitions in the cooking domain, we have implemented a system to detect new ingredients in the input by searching the nouns in the lexical database Wordnet. Wordnet is a lexical database in English, developed at Princeton University under the direction of George A. Miller. This lexical database allows searching a concept to get the definition, synonyms and antonyms, among other things.

For every noun in the query that is also in Wordnet, is a candidate to be an ingredient, the system checks if it is already included the ontology. If the ingredient is not in the ontology, we try to automatically include and classify it in the ontology. Using WordNet the application checks whether the definition of that noun is feasible to belong to any of the categories present in the ontology, if it is, JADAWeb adds it to the system memory, if the system cannot select a unique branch, it asks the user to select the concept to allocate the ingredient.

3.3 Algorithm definition

JADAWeb includes the capability of classifying a new ingredient in the ontology using the information from Wordnet.

Our algorithm first searches an ingredient in the ontology using its name and also using all the synonyms that Wordnet gives for this concept in the same synset. If any of the synonyms belongs to the ontology, the search is done. If not, the system looks for the best node to attach the ingredient.

The parser returns two lists of ingredients. A first list with the ingredients that the user wants to use. And a second list with the ingredients that the user

wants to avoid. The ontology update algorithm proceeds as follows: it takes each element of those lists and checks if the item is already included in the ontology. If the item or any of those synonyms are registered then the algorithm is done; if the ingredient does not appear then the system searches the best node to put the new ingrediente, but if the best node is not found, the user guides the system to include the ingredient in the ontology. A branch is defined as the part of the tree whose items have a predecessor or successor relationship to each other. A child is the successor of his parent, grandparents ... and the parent predecessor of children, grandchildren (see Figure 3). To decide the best branch to introduce an ingredient in the ontology, the system uses the word definition in Wordnet. As only nouns are feasible to be ingredients, the system discards the other parts of speech.

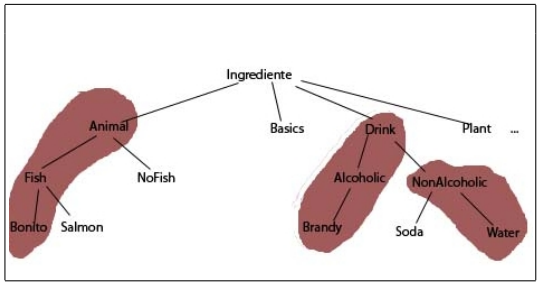

Fig. 3. Fragment of the ontology using by JadaWeb, using two examples of branches within the tree.

All the nouns in the text of the candidate ingredient definition are searched in the ontology.

If every noun in the definition appears in the same branch, then a candidate predecessor to allocate the new ingredient would be the lowest item of the branch that is not a leaf. If it is a leaf, the new ingredient will be attached to the predecessor of the leaf.

If the definition includes words that are in different branches, each branch is checked. The system will select the branch which has more nouns of the definitions of the new ingredient. If two or more branches have the same number of ingredients, the item would have a number of potential predecessor (remember that the system is searching for the best predecessor to hang the new ingredient). This is the only case in which both predecessors are returned to the user. The remaining cases, we obtain a unique solution of the problem, although not necessarily the best.

In this way, the system puts the ingredient as a child of a node according its definition and synonyms. Example: We have tested the algorithm with Shark (the parent the system recomends is fish), saffron (the parents the system recomends ar flower and citrus), yeast (cruciferous) and others, and we obtain a very satisfactory result in an 85 % of the new ingredients included.

3.4 Cases

In this new version of the system we have used the case base that is provided by the organizers of the CCC-10, which is the same than for the CCC-09. The recipe cases are organized in an XML file that is processed using SAX and DOM to extract the information to load it in the system memory.

3.5 Non wanted ingredients treatment

Previous versions of JADACook only remove the ingredients that the user does not want. JADAWeb includes a new algorithm for removing the ingredients that shares the same branch in the tree with the non wanted ingredients. This option is good when the user is not very specific. However, if the user is very specific, or (s)he does not want a special ingredient, our algorithm could remove other ingredients that the user might want. For example, if the user specifies (s)he does not want *sardine*, our system will remove the term *sardine* and other kind of similar fishes like *anchovies* or *herring*.

4 Case-Based reasoning process

4.1 JadaCook2

JADAWeb inherits the similarity function of JADACook2[2], and adds some functionality to it. The original similarity function of JADACook2 works as follows: first, the system takes the ingredients for the wish list of the user, and it searches for recipes that contain these ingredients. If a recipe contains an ingredient it increases by 1.0 the similarity value. The similarity value is normalized by the number of ingredients in the query.

During adaptation JadaCook2 does not reject an ingredient when it is not in the recipe. When it happens, the system searches for ingredients that share the same branch of the original ingredient in the ontology, if it is feasible the algorithm increase by 0.8 the similarity value of this recipe. JadaCook2 searches in all recipes contained by the case base. Finally, the system gets a list of recipes with the information of how similar is each one in according to the ingredients introduced by the user.

The user can also specify in the query the type of meal (s)he wants, the type of diet, the type of cuisine and if (s)he wants some dietary practices. The system adds some ingredients if the user wants a specific type of meal, i.e. if the user selects a *chinese* meal the system will add some ingredients like rice, bamboo, carrot ,ginger, chicken, duck, pig, pork or soy sauce.

4.2 JADAWeb

JADAWeb includes new features. The first one is the ability to look for ingredients to adapt not only among the siblings, but also going up in the hierarchy. The similarity function has the same criteria as the degree of consanguinity of

two people. Two siblings have a similarity value of 0.6, because both are descendent of the same parents. Therefore, a rise of level, subtract 0.4 of similarity. For two cousins, subtract 0.8: 0.4 because of the rise to the parent and other 0.4 because of the climb to the grandparent. Going down in the level, not penalize. Thus, another example: the children of a sibling of an ingredient have the same similarity as the sibling (0.6), the similarity of a cousin's children are the same as the cousin and also the same as any of the grandfather's descendants (which do not fall for the same branch of the one that is being compared) which is 0.2 in this case. In this way, you can use other similar ingredients. The number of levels can be parameterized and the default value is 2, i.e, the system only searches among siblings and their descendants.

Table 1. Some examples of the fuzzy table.

Ingredient Origin	**Ingredient targez**	**Value**
Rice	Macaroon	0.4
Stock	Water	0.9
Stock	Bouillon cubes	0.9
Milk	Cream	0.6
Milk	Water	0.6

Another novelty in JadaWeb is a table of similarity between fuzzy ingredients. It is a table with specific values (fuzzy) that gives meaning to say that rice is replaceable by macaroni with a value of 0.4 to add to the similarity function. All these pairs of ingredients are defined in an XML file which indicates what ingredients are similar and the associated fuzzy value. The fuzzy table is not only used to replace pairs of ingredients. The system also can decide replacing one ingredient by a group of similar ingredients: stock instead of water and bouillon cubes with a similarity of 0.9, milk instead of water and cream in a similarity value of 0.6.

4.3 Measure evaluation of the similarity function.

We have proposed the following experiment as a measure for evaluating the similarity function of the system.

We made a survey with a set of people(12), each member tested the system with 5 JADAWeb queries. Each user has introduced his/her a query in natural language in English and we gave them the choice of which level they want. The tests were carried out with the three levels of adaptation of ingredients as shown in Section 4.2. After the CBR process the user evaluates the system results telling us if the output of the system is satisfactory for him/her, both the recipes and the order. Table 2 shows the results of the survey.

Table 2. Results obtained from the measure of evaluation

Similarity level	Satisfied user	Substitutions
1	90.0%	5.0%
2	70.0%	10.0%
3	40.0%	30.0%

If the similarity level increases then the number of substitutions also increases. With a similarity level that allows only adaptation between ingredients that share the same branch (similarity level = 1), the satisfaction with the given recipe is very high. However, the result is worse when we use higher levels. This is because people do not expect those adapted elements. In various dishes (e.g., replacing the lemon with bread), the result is quite concerned, but we could see that a higher-level change (similarity level $\geq$ 2) makes the user obtain another recipe recommended that he or she could not expect it. With a deeperd ontology, higher level adjustments would score much more satisfactory results. The result of 70 % for level 2 is good, but the 40 % of level 3 is very low.

5 Results and examples

In this section we show a list of examples of the JADAWeb system. For each query we describe the recipe proposed by the system in the first place. However, for each query the system returns five suggestions ordered by the similarity value.

First Experiment In this experiment we have introduced the list of ingredients for a typical Spanish recipe: paella. We have not specified any type of diet, cuisine or dish and there are no dietary practices.

Q1: "I want to eat rice, saffron, shrimps, chicken, crab, squid but I don't like sugar".
Answer: the first answer is Paella (Paella in a Pot) that contains many of the ingredients contained in the query.

Second Experiment In this experiment, we write a query to obtain any type of pizza with ham, tomato and cheese, and the pizza must not contain onion. We do not specify any type of diet or type of cuisine, but we introduce a type of dish (pizza). Although the term "pizza" is written in the main query in natural language, the system will recognize it as a type of dish.

Q2: "I like pizza with ham, cheese and tomato but I don't want onions".

Answer: The system returns some types of pizza ("Pizza Quiche", "Idiot Bread Pizza", "Breakfast Pizza", "Fruit Pizza" and "Pineapple Cream Cheese

Pizza") which contains ham and cheese or cheese and tomato. We also obtain some pizzas with ingredients adapted changing tomato with pepper and cheese with egg, penalizing in 0.4 each one.

Third Experiment In this experiment, we tested the system with a fruit salad. We do not specify any type of diet or type of cuisine but we specify that the type of dish is a salad.

Q2: "I like to eat some fruit, like orange, lemon and apple but I don't want kiwi".
Answer: The system returns five salads: first a salad that contains a variety of fruits: "Portable Salad Dressing" in addition to other salads as "Creamy Waldorf Salad".

Comparing JADAWeb with Cookiis As a measure of the behaviour of our system, we have compared JADAWeb results with last year winner: CookIIS[7]. We show the similarities and differences between the two systems by comparing the retrieved sets of recipes of both systems.

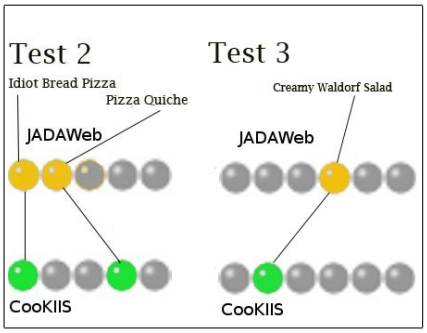

Fig. 4. Comparation with test 2.

In the first query there is no coincidence between the retrieved recipes. JADAWeb retrieves *Paella* in the first place that is a highlight answer because we used the paella ingredients to make this query. Cookiis obtains *Pad Thai* in the first place. In the other queries we have some coincident results between the first 5 suggested recipes. In query 2 the two systems retrieve two common recipes, although in a different order. *Idiot Bread Pizza* is the first option in both. Pizza Quiche is the second in JADAWeb, but the fourth in Cookiis. In query 3 we only have one coincident result: Creamy Waldorf Salad.

6 Conclusions

JADAWeb system improves and extends the JADACook2 [2] system. Its web-based interface gives a broader access to the system and Web accessibility. Accomplishing the requirements of AA or AAA W3C makes the system useful for any person although the person has an inability, i.e blindness.

By using a textual input, we enable a more natural treatment improvement in the interaction with the final user. We can suggest some future work by using a system that allows voice recognition that could be connected with our dependency parser.

JADAWeb extends the possibilities by incorporating a fuzzy table which allows similarity between two distant elements or types in the ontology, and the inclusion of various levels in the adaptation algorithm.

It should be noted the great contribution obtained by using WordNet, it contributes allowing more synonyms for some terms and it extends largely the knowledge that it has, avoiding the limitation of the use of a finite ontology of ingredients.

References

1. Herrera, P.J., Iglesias, P., Romero, D., Rubio, I., Díaz-Agudo, B.: Jadacook: Java application developed and cooked over ontological knowledge. In Schaaf, M., ed.: ECCBR Workshops. (2008) 209–218
2. Herrera, P.J., Iglesias, P., Sánchez, A.M.G., Díaz-Agudo, B.: Jadacook 2: Cooking over ontological knowledge. In: ICCBR Workshops. Computer Cooking Contest. (2009)
3. Recio-García, J.A., D'iaz-Agudo, B., González-Calero, P.A.: Boosting the performance of cbr applications with jcolibri. In: ICTAI, IEEE Computer Society (2009) 276–283
4. Fields, D.K., Kolb, M.A., Bayern, S.: Web Development with Java Server Pages. Manning Publications Co., Greenwich, CT, USA (2001)
5. Buchholz, S., Marsi, E.: Conll-x shared task on multilingual dependency parsing. In: Proceedings of the 10th Conference on Computational Natural Language Learning (CoNLL–X). (2006) 149–164
6. Fellbaum, C., ed.: WordNet: an electronic lexical database. MIT Press (1998)
7. Hanft, A., Ihle, N., Bach, K., Newo, R.: Cookiis - competing in the first computer cooking contest. KI **23** (2009) 30–33
8. Lin, D.: Dependency-based evaluation of minipar. In: Proc. Workshop on the Evaluation of Parsing Systems. Granada (1998)
9. Nivre, J., Hall, J., Nilsson, J.: Maltparser: A data-driven parser-generator for dependency parsing. In: In Proc. of LREC-2006. (2006) 2216–2219
10. Sampson, G.: (Briefly noted english for the computer: The susanne corpus and analytic scheme)