Improving Ingredient Substitution using Formal Concept Analysis and Adaptation of Ingredient Quantities with Mixed Linear Optimization

Emmanuelle Gaillard, Jean Lieber, and Emmanuel Nauer

Université de Lorraine, LORIA — 54506 Vandœuvre-lès-Nancy, France
CNRS — 54506 Vandœuvre-lès-Nancy, France
Inria — 54602 Villers-lès-Nancy, France
`firstname.lastname@loria.fr`

Abstract. This paper presents the participation of the TAAABLE team to the 2015 Computer Cooking Contest. The TAAABLE system addresses the *mixology* and the *sandwich* challenges. For the *mixology challenge*, the 2014 TAAABLE system was extended in two ways. First, a formal concept analysis approach is used to improve the ingredient substitution, which must take into account a limited set of available foods. Second, the adaptation of the ingredient quantities has also been improved in order to be more realistic with a real cooking setting. The adaptation of the ingredient quantities is based on a mixed linear optimization. The team also applied TAAABLE to the *sandwich challenge*.

Keywords: case-based reasoning, formal concept analysis, adaptation of ingredient quantities, mixed linear optimization.

1 Introduction

This paper presents the participation of the TAAABLE team to the *mixology* and to the *sandwich* challenges of the 2015 Computer Cooking Contest (CCC). The TAAABLE system is based on many methods and techniques in the area of knowledge representation, knowledge management and natural language processing [1]. Currently, it is built over TUUURBINE (`http://tuuurbine.loria.fr`), a generic case-based reasoning (CBR) system over RDFS [2] which allows reasoning over knowledge stored in a RDF store, as the one provided by the contest.

For this edition of the CCC, TAAABLE has been extended in order to improve the ingredient substitution procedure which must manage unavailable foods. An approach based on formal concept analysis (FCA) allows improving ingredient substitutions. Moreover, the adaptation of the ingredient quantities has also been improved in order to be more realistic with a real cooking setting. The adaptation of the ingredient quantities is based on mixed linear optimization. This adaptation takes into account the preference unit given in the source recipe and proposes quantities which are usual. For example, when the ingredient is a lemon, its quantity will take the form of a human easy understandable value (i.e. a quarter, a half, etc. instead of *54 g*, which corresponds to a half lemon).

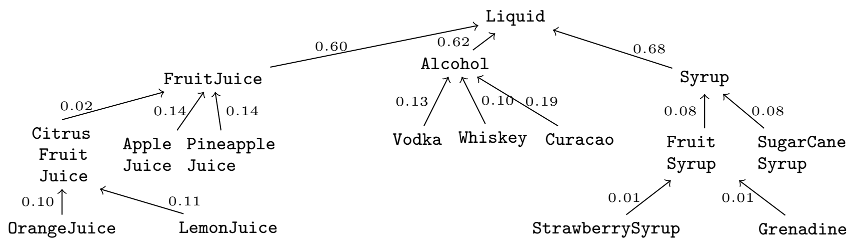

Fig. 1. The hierarchy forming the domain knowledge used in the running example with the generalization costs used as retrieval knowledge.

Section 2 introduces the core of the TAAABLE system. Section 3 details the new approaches developed specially for the mixology challenge. Section 4 explains the system submitted for the sandwich challenge.

2 The TAAABLE system

The challenges, proposed by the CCC since its first edition consists in proposing, according to a set of initial recipes, one or more recipes matching a user query composed of a set of wanted ingredients and a set of unwanted ingredients. The TAAABLE system addresses this issue through an instantiation of the generic CBR TUUURBINE system [3], which implements a generic CBR mechanism in which adaptation consists in retrieving similar cases and in replacing some features of these cases in order to adapt them as a solution to a query.

2.1 TUUURBINE founding principles

TUUURBINE is a generic CBR system over RDFS. The domain knowledge is represented by an RDFS base `DK` consisting of a set of triples of the form $\langle C\ \texttt{subClassOf}\ D\rangle$ where C and D are classes which belong to a same hierarchy (e.g, the food hierachy). Fig. 1 represents the domain knowledge for the running examples by a hierarchy whose edges $C \xrightarrow{x} D$ represent the triples $\langle C\ \texttt{subClassOf}\ D\rangle$. The retrieval knowledge is encoded by a cost function: $\texttt{cost}(\langle C\ \texttt{subClassOf}\ D\rangle) = x$ for an edge $C \xrightarrow{x} D$. This cost can be understood intuitively as the measure of "the generalization effort" from C to D. How this cost is computed is detailed in [1].

A TUUURBINE case `case` is described by a set of triples of the form $\langle URI_{\texttt{case}}\ \texttt{prop}\ \texttt{val}\rangle$, where $URI_{\texttt{case}}$ is the URI of `case`, `val` is either a resource representing a class of the ontology or a value and `prop` is an RDF property linking `case` to a hierarchy class or to the value. For simplification, in this paper, we represent a case by a conjunction of expressions only of the form `prop` : `val`. For example, the "Rainbow" recipe is represented by the following index `R`, which means that "Rainbow" is a cocktail recipe made from vodka, orange juice, grenadine and curacao (`ing` stands for *ingredient*).

$$\begin{aligned}\texttt{R} = {} & \texttt{dishType}:\texttt{CocktailDish} \\ & \wedge \texttt{ing}:\texttt{Vodka} \wedge \texttt{ing}:\texttt{OrangeJuice} \wedge \texttt{ing}:\texttt{Grenadine} \wedge \texttt{ing}:\texttt{Curacao}\end{aligned} \tag{1}$$

For instance, the first conjunct of this expression means that the triple $\langle URI_R$ `dishType CocktailDish`$\rangle$ belongs to the knowledge base.

2.2 Tuuurbine query

A Tuuurbine query is a conjunction of expressions of the form $\texttt{sign}\,\texttt{prop}:\texttt{val}$ where $\texttt{sign} \in \{\epsilon, +, !, -\}$, `val` is a resource representing a class of the ontology and `prop` is an RDF property belonging to the set of properties used to represent cases. For example,

$$\begin{aligned}\texttt{Q} = {} & +\texttt{dishType}:\texttt{CocktailDish} \\ & \wedge \texttt{ing}:\texttt{Vodka} \wedge \texttt{ing}:\texttt{Grenadine} \wedge !\texttt{ing}:\texttt{Whiskey}\end{aligned} \tag{2}$$

is a query to search "a cocktail with vodka and grenadine syrup but without whiskey".

The signs ϵ (*empty sign*) and $+$ are "positive signs": they prefix features that the requested case must have. $+$ indicates that this feature must also occur in the source case whereas ϵ indicates that the source case may not have this feature, thus the adaptation phase has to make it appear in the final case.

The signs ! and $-$ are "negative signs": they prefix features that the requested case must not have. $-$ indicates that this feature must not occur in the source case whereas ! indicates that the source case may have this feature, and, if so, that the adaptation phase has to remove it.

2.3 Tuuurbine retrieval process

The retrieval process consists in searching for cases that best match the query. If an exact match exists, the corresponding cases are returned. For the query `Q` given in (2), the "Rainbow" recipe is retrieved without adaptation. Otherwise, the query is relaxed using a generalization function composed of one-step generalizations, which transforms `Q` (with a minimal cost) until at least one recipe of the case base matches $\Gamma(Q)$.

A one step-generalization is denoted by $\gamma = \texttt{prop} : \texttt{A} \rightsquigarrow \texttt{prop} : \texttt{B}$, where `A` and `B` are classes belonging to the same hierarchy with $\texttt{A} \sqsubseteq \texttt{B}$, and `prop` is a property used in the case definition. This one step-generalization can be applied only if `A` is prefixed by ϵ or ! in `Q`. If `A` is prefixed by !, thus `B` is necessarily the top class of the hierarchy. For example, the generalization of $!\texttt{ing} : \texttt{Rum}$ is $\epsilon\texttt{ing} : \texttt{Food}$, meaning that if rum is not wanted, it has to be replaced by some other food. Classes of the query prefixed by $+$ and $-$ cannot be generalized.

Each one-step generalization is associated with a cost denoted by $\texttt{cost}(A \rightsquigarrow B)$. The generalization Γ of `Q` is a composition of one-step generalizations γ_1, $\ldots \gamma_n$: $\Gamma = \gamma_n \circ \ldots \circ \gamma_1$, with $\texttt{cost}(\Gamma) = \sum_{i=1}^{n} \texttt{cost}(\gamma_i)$. For example, for:

$$\begin{aligned}\texttt{Q} = {} & +\texttt{dishType}:\texttt{CocktailDish} \\ & \wedge \texttt{ing}:\texttt{Vodka} \wedge \texttt{ing}:\texttt{PineappleJuice} \wedge \texttt{ing}:\texttt{Grenadine} \wedge !\texttt{ing}:\texttt{Whiskey}\end{aligned} \tag{3}$$

`PineappleJuice` is relaxed to `FruitJuice` according to the domain knowledge of Fig. 1. At this first step of generalization, $\Gamma(\texttt{Q}) =$

`dishType:CocktailDish`$\wedge$`ing:Vodka`$\wedge$`ing:FruitJuice`$\wedge$`!ing:Whiskey`, which matches the recipe described in (1), indexed by `OrangeJuice`, a `FruitJuice`.

2.4 Tuuurbine adaptation process

When the initial query does not match existing cases, the cases retrieved after generalization have to be adapted. The adaptation consists of a specialization of the generalized query produced by the retrieval step. According to $\Gamma(\mathtt{Q})$, to `R`, and to `DK`, the ingredient `OrangeJuice` is replaced with the ingredient `PineappleJuice` in `R` because `FruitJuice` of $\Gamma(\mathtt{Q})$ subsumes both `OrangeJuice` and `PineappleJuice`. Tuuurbine implements also an adaptation based on rules where some ingredients are replaced with others in a given context [4]. For example, in cocktail recipes, replacing `OrangeJuice` and `StrawberrySyrup` with `PineappleJuice` and `Grenadine` is an example of an adaptation rule. This rule-based adaptation is directly integrated in the retrieval process by searching cases indexed by the substituted ingredients for a query about the replacing ingredients, for example by searching recipes containing `OrangeJuice` and `StrawberrySyrup` for a query about `PineappleJuice` and `Grenadine`.

2.5 Taaable as a Tuuurbine instantiation

The Taaable knowledge base is WikiTaaable (`http://wikitaaable.loria.fr/`), the knowledge base made available for this CCC edition. WikiTaaable is composed of the four classical knowledge containers: (1) the domain knowledge contains an ontology of the cooking domain which includes several hierarchies (about food, dish types, etc.), (2) the case base contains recipes described by their titles, the dish type they produce, the ingredients that are required, the preparation steps, etc., (3) the adaptation knowledge takes the form of adaptation rules as introduced previously, and (4) the retrieval knowledge, which is stored as cost values on subclass-of relations and adaptation rules.

In WikiTaaable, all the knowledge (cases, domain knowledge, costs, adaptation rules) is encoded in a triple store, because WikiTaaable uses Semantic Media Wiki, where semantic data is stored into a triple store. So, plugging Tuuurbine over the WikiTaaable triple store is quite easy because it requires only to configure Tuuurbine by giving the case base root URI, the ontology root URI and the set of properties on which reasoning may be applied.

3 Mixology challenge

The mixology challenge consists in retrieving a cocktail that matches a user query according to a set of available foods given by the CCC organizers (white rum, whiskey, vodka, orange juice, pineapple juice, sparkling water, coca-cola, beer grenadine syrup, lemon juice, mint leaves, lime, ice cube, brown sugar, salt, and pepper). Tuuurbine queries can express this kind of request using the ϵ and ! prefixes. Section 3.1 explains how the user query is transformed to take into account only the available foods, before being submitted to Tuuurbine. Two additionnal processes have been implemented to improve the Tuuurbine adaptation result. The first process searches, when some ingredients of the source recipe are not available, the best way to replace them, or in some cases, to remove them (see Section 3.2). The second process uses Revisor/CLC (see Section 3.4) to adapt quantities. A new formalization of the quantity adaptation problem is proposed to obtain more realistic quantity values, taking into account the type of unit given in the source case (see Section 3.4).

3.1 Query building

For the mixology challenge, where an answer must only contain the available food, the query may be built by adding to the initial user query the minimal set of classes of the food hierarchy that subsume the set of foods which are not available, each class being negatively prefixed by !. For example, let us assume that `OrangeJuice` and `PineappleJuice` are the only available fruit juices, that `Vodka` and `Whiskey` are the only available alcohols, that `SugarCaneSyrup` and `Grenadine` are the only available syrups, and that the user wants a cocktail recipe with `Vodka` but without `SugarCaneSyrup`. The initial user query will be $\texttt{Q} = +\texttt{dishType}:\texttt{CocktailDish} \wedge \epsilon\texttt{ing}:\texttt{Vodka} \wedge \texttt{!ing}:\texttt{SugarCane}$. According to Fig. 1, `LemonJuice`, `AppleJuice`, `Curacao`, and `StrawberrySyrup` will be added to this initial query with a ! for expressing that the result cannot contain one of these non available classes of food, which includes their descendant classes. The extended query `EQ` submitted to Tuuurbine will be:

$$\begin{aligned}\texttt{EQ} = \texttt{Q} &\wedge \texttt{!ing}:\texttt{LemonJuice} \wedge \texttt{!ing}:\texttt{AppleJuice}\\ &\wedge \texttt{!ing}:\texttt{StrawberrySyrup} \wedge \texttt{!ing}:\texttt{Curacao}\end{aligned}$$

For this example, Tuuurbine retrieves the “Rainbow” recipe with the adaptation “replace `Curacao` with `Food`”, due to `!ing:Curacao`.

In order to replace `Curacao` by something more specific than `Food`, a new approach based on FCA is proposed.

3.2 Using FCA to search the best ingredient substitution

When ingredients of the source case must be replaced because these pieces of food are not available, we choose FCA to exploit ingredient combination in cocktail recipes in order to search which ingredient(s) is/are the most used with the ones already used in the recipe that must be adapted. FCA is a classification method allowing object grouping according to the properties they share [5]. FCA takes as input a *binary context*, i.e. a table in which objects are described by properties. Table 1 shows an example of binary context with 7 objects (which are cocktails), described by two kinds of properties: the ingredients they use, and some more generic ingredient classes: `_Alcohol`, the generic class of recipes with at least one alcohol, and `_Sugar`, the generic class of recipes with at least one sweet ingredient, like sugar or syrup. These generic classes are prefixed by _ to be distinguished from the concrete ingredients. For example, the object `Screwdriver` has the properties `Vodka` and `Orange juice` (the ingredients used in this cocktail), and `_Alcohol`, because `Vodka` is an alcohol.

FCA produces *formal concepts* as output. A formal concept is a pair (I, E) where I is a set of properties, E is a set of objects, respectively called the *intent*

	_Alcohol	Vodka	White rum	Tequila	Cacha ca	Blue cucacao	Orange juice	Coca-cola	Lime	_Sugar	White sugar	Cane sugar syrup	Grenadine
Screwdriver	×	×					×						
Rainbow	×	×				×	×			×			×
Tequila sunrise	×			×			×			×			×
Ti′Punch	×		×						×	×		×	
Daiquiri	×		×						×	×		×	
Caipirinha	×				×				×	×	×		
Cuba libre	×		×					×	×				

Table 1. A binary context for cocktails, described by their ingredients and two generic food classes (`_Alcohol` and `_Sugar`).

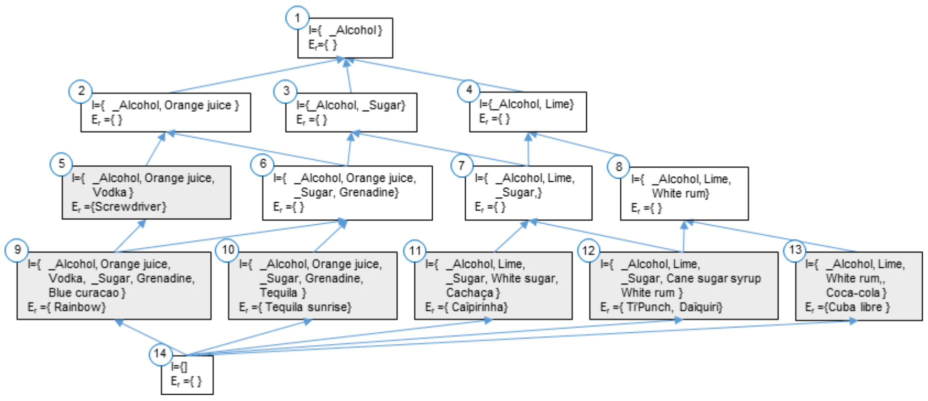
1
I={ _Alcohol}
E_r={ }
2
I={ _Alcohol, Orange juice }
E_r ={ }
3
I={_Alcohol, _Sugar}
E_r ={ }
4
I={_Alcohol, Lime}
E_r ={ }
5
I={ _Alcohol, Orange juice, Vodka }
E_r ={Screwdriver}
6
I={ _Alcohol, Orange juice, _Sugar, Grenadine}
E_r ={ }
7
I={ _Alcohol, Lime, _Sugar,}
E_r ={ }
8
I={ _Alcohol, Lime, White rum}
E_r ={ }
9
I={ _Alcohol, Orange juice, Vodka, _Sugar, Grenadine, Blue curacao }
E_r ={ Rainbow}
10
I={ _Alcohol, Orange juice, _Sugar, Grenadine, Tequila }
E_r ={ Tequila sunrise}
11
I={ _Alcohol, Lime, _Sugar, White sugar, Cachaça }
E_r ={ Caipirinha}
12
I={ _Alcohol, Lime, _Sugar, Cane sugar syrup White rum }
E_r ={ TiPunch, Daiquiri}
13
I={ _Alcohol, Lime, White rum,, Coca-cola }
E_r ={Cuba libre }
14
I={}
E_r ={ }

Fig. 2. Concept lattice organizing cocktails according to their ingredients.

and the *extent* of the formal concept, such that (1) I is the set of all properties shared by the objects of E and (2) E is the set of all objects sharing properties in I. The formal concepts can be ordered by extent inclusion, also called *specialisation* between concepts, into what is called a *concept lattice*. Fig. 2 illustrates the lattice corresponding to the binary context given in Table 1. On this figure, the extents E are given through a reduced form (noted E_r): the objects appear in the most specific concepts, the complete extent can be computed by the union of objects belonging to the subconcepts. So, the top concept (#1, in the figure) contains all the objects. In our example, its intent is `_Alcohol`, a property shared by all the objects. By contrast, the bottom concept is defined by the set of all properties. In our example, its extent is empty as none of the objects are described by all the properties.

To search a replacing ingredient in a given recipe or in a recipe according to pieces of food that will be kept, the idea is to exploit the lattice which captures concept similarities and organization. For example, concept #7, which intent is {`_Alcohol`, `Lime`, `_Sugar`}, allows an access to 3 cocktails containing at least one alcohol, at least one sugar, and lime. Adapting a cocktail can be based on the closeness between concepts. For example, when a replacing ingredient is searched

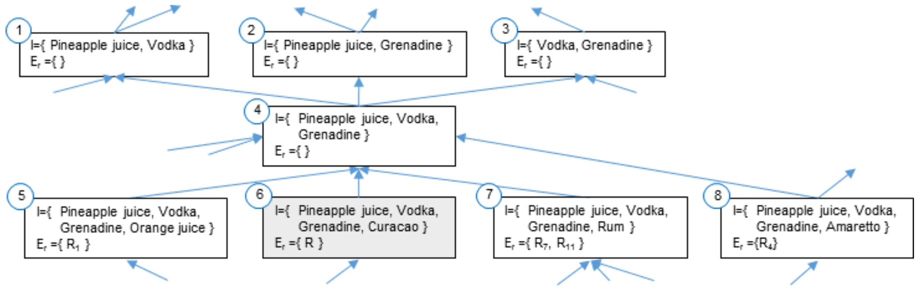
1
I={ Pineapple juice, Vodka }
E_r ={ }
2
I={ Pineapple juice, Grenadine }
E_r ={ }
3
I={ Vodka, Grenadine }
E_r ={ }
4
I={ Pineapple juice, Vodka, Grenadine }
E_r ={ }
5
I={ Pineapple juice, Vodka, Grenadine, Orange juice }
E_r ={ R_1 }
6
I={ Pineapple juice, Vodka, Grenadine, Curacao }
E_r ={ R }
7
I={ Pineapple juice, Vodka, Grenadine, Rum }
E_r ={ R_7, R_{11} }
8
I={ Pineapple juice, Vodka, Grenadine, Amaretto }
E_r ={R_4}

Fig. 3. Part of the concept lattice built from recipes using `PineappleJuice`, `Vodka`, and `Grenadine` (the ingredients that will be used in the resulting cocktail).

for *Cachaça* in the *Caipirinha* cocktail (in the intent of concept #11), some similar concepts (i.e. sharing a same super-concept) can be used. In the lattice given in example, concept #11 can be generalized to concept number #7, which extent contains cocktails with some alcohol, lime and some sugar. The cocktails in the extent of concept #12 are similar to the one of concept #11, because they share the `_Alcohol`, `Lime`, and `_Sugar` properties. When removing *"Cachaça"* from the *Caipirinha*, a possible ingredient for substitution, given by the lattice, could be `White rum`.

The approach exploiting the link between the concepts is used in many works using FCA for information retrieval. In Carpineto and Romano [6], the documents which are good answers to a query are searched in the lattice built from the document properties and from the query, around the concept representing the query. The same authors use this neighbour relation between concepts in a lattice for ordering documents returned by an information retrieval system [7].

Let C_R be the formal concept such that $E_r(C_R) = \{R\}$. A formal concept C close to C_R is searched according the following procedure. C is such that its intent $I(C)$ does not contain the substituting ingredient (`Curacao` in the example) and maximizes $|E_r(C)|$. First, C is searched in the ascendants of C_R, then in its siblings, and finally in the descendants of the siblings. The ingredient to be substituted is replaced by $I(C) \setminus I(C_R)$.

3.3 Real example of food substitution using FCA

To implement our approach, data about ingredient combinations in cocktail recipes has been collected. For this, we queried *Yummly* (`http://www.yummly.com/`). 16 queries were submitted; each query was composed of one ingredient (one available food) and was parametered to return all the *Yummly* cocktails and beverage recipes containing this ingredient. 9791 recipes have been collected. Unfortunately, the *Yummly* search engine does not necessarily return answers satisfying the query. So, the results are filtered, only to keep recipes that use at least one available food. Afterwards, the remaining recipes are deduplicated. After filtering and deduplicating, 6114 recipes are available, but only 1327 of them combine at least 2 available foods.

We show now, with query (3), how, after proposing to replace `OrangeJuice` with `PineappleJuice` and `StrawberrySyrup` with `Grenadine` in `R`, TAAABLE searches to replace `Curacao` which is not in the set of available foods. A part of the lattice resulting from the binary table containing recipes with `PineappleJuice`, `Grenadine` and `Vodka` is given in Fig. 3. Concept #6 corresponds to `R`, the recipe that must be adapted, and which has been added in the binary table to appear in the lattice. The most similar ingredient combination which includes `PineappleJuice`, `Grenadine` and `Vodka` is given by concept #7. Indeed, concept #8 cannot be used to produce a substitution because its intent contains `Amaretto` which is not an available food. Concept #5 intent contains `OrangeJuice`, an available food, but concept #5 is less close to concept #6 than concept #7, according to the selection procedure based on the maximal number of objects of E_r.

3.4 Adaptation of quantities with mixed integer linear optimization

Let us consider the following adaptation problem:

`Source` =

Recipe "Eggnog" (10 glasses)
10 cℓ of armagnac, 25 cℓ of rum, half a liter of milk, 5 eggs, 125 g of granulated sugar, 25 cℓ of fresh cream

`Q` = "I want a cocktail recipe with cream but without egg or armagnac."

for which TUUURBINE produces the following ingredient substitution:

substitute egg and armagnac **with** banana and kirsch (4)

It must be noticed that this example does not comply with the constraints of the cocktail challenge (banana is not an available food), but has been chosen in order to illustrate various ideas related to adaptation of quantities. The approach to ingredient quantity adaptation is based on belief revision [8], applied to a formalization suited to adaptation of quantities. First, the adaptation problem (`Source`, `Q`) and the domain knowledge `DK` are formalized. Then, this adaptation process is described.

Formalization. Numerical variables are introduced to represent the ingredient quantities in a recipe. For the example, the following variables are introduced, for each food class `C`: $\mathtt{alcohol}_\mathtt{C}$, $\mathtt{mass}_\mathtt{C}$, $\mathtt{number}_\mathtt{C}$, $\mathtt{sugar}_\mathtt{C}$ and $\mathtt{volume}_\mathtt{C}$, which represent, respectively, the quantity (in grams) of alcohol in the ingredient `C` of the recipe, its mass (in grams), its number, its quantity (in grams) of sugar and its volume (in centiliters).[1] Therefore, the retrieved recipe can be expressed in this formalism by:

$$\begin{aligned}\mathtt{Source} = {} & (\mathtt{volume}_\mathtt{Armagnac} = 10) \wedge (\mathtt{volume}_\mathtt{Rum} = 25) \wedge (\mathtt{volume}_\mathtt{Milk} = 50) \\ & \wedge (\mathtt{number}_\mathtt{Egg} = 5) \wedge (\mathtt{mass}_\mathtt{GranulatedSugar} = 125) \\ & \wedge (\mathtt{volume}_\mathtt{FreshCream} = 25)\end{aligned} \tag{5}$$

[1] One could consider other variables, e.g., the calories of ingredients, which would make possible to add constraints on the total number of calories in a dish.

In theory, all the variables could be continuous (represented by floating-point numbers). However, this can lead to adapted cases with, e.g., $\mathtt{number}_{\mathtt{Egg}} = 1.7$, which is avoided in most recipe books! For this reason, some variables v are declared as integer (denoted by $\tau(v) = \mathtt{integer}$), the other ones as real numbers (denoted by $\tau(v) = \mathtt{real}$).

The domain knowledge DK consists of a conjunction of *conversion equations*, *conservation equations* and *sign constraints*. The following conversion equations state that one egg without its shell has (on the average) a mass of 50 g, a volume of 5.2 cℓ, a quantity of sugar of 0.77 g and no alcohol:

$$\begin{aligned} \mathtt{mass}_{\mathtt{Egg}} &= 50 \times \mathtt{number}_{\mathtt{Egg}} & \mathtt{volume}_{\mathtt{Egg}} &= 5.2 \times \mathtt{number}_{\mathtt{Egg}} \\ \mathtt{sugar}_{\mathtt{Egg}} &= 0.77 \times \mathtt{number}_{\mathtt{Egg}} & \mathtt{alcohol}_{\mathtt{Egg}} &= 0. \end{aligned} \tag{6}$$

with $\tau(\mathtt{mass}_{\mathtt{Egg}}) = \tau(\mathtt{volume}_{\mathtt{Egg}}) = \tau(\mathtt{sugar}_{\mathtt{Egg}}) = \tau(\mathtt{alcohol}_{\mathtt{Egg}}) = \mathtt{real}$ and $\tau(\mathtt{number}_{\mathtt{Egg}}) = \mathtt{integer}$.

The following equations are also conjuncts of DK and represent the conservation of masses, volumes, etc.:

$$\begin{aligned} \mathtt{mass}_{\mathtt{EggOrEquivalent}} &= \mathtt{mass}_{\mathtt{Egg}} + \mathtt{mass}_{\mathtt{Banana}} \qquad (7) \\ \mathtt{volume}_{\mathtt{Food}} &= \mathtt{volume}_{\mathtt{Liquid}} + \mathtt{volume}_{\mathtt{SolidFood}} \\ \mathtt{volume}_{\mathtt{Liquid}} &= \mathtt{volume}_{\mathtt{Brandy}} + \mathtt{volume}_{\mathtt{Rum}} + \mathtt{volume}_{\mathtt{FreshCream}} + \ldots \\ \mathtt{volume}_{\mathtt{Brandy}} &= \mathtt{volume}_{\mathtt{Armagnac}} + \mathtt{volume}_{\mathtt{Kirsch}} + \ldots \end{aligned}$$

where Food is the class of the food (any ingredient of a recipe is an instance of Food) and, e.g., $\mathtt{alcohol}_{\mathtt{Rum}}$ is related to $\mathtt{volume}_{\mathtt{Rum}}$ thanks to the conversion equation $\mathtt{alcohol}_{\mathtt{Rum}} = 0.4 \times \mathtt{volume}_{\mathtt{Rum}}$. Actually, equation (7) corresponds to the substitution of eggs by bananas.

Such conservation equations can be acquired using parts of the food hierarchy, thanks to some additional information. For instance, if C is a class of the hierarchy and $\{D_1, D_2, \ldots, D_p\}$ is a set of subclasses of C forming a partition of C (i.e., for each individual x of C, there is exactly one $i \in \{1, 2, \ldots, p\}$ such that x belongs to D_i), then $\mathtt{mass}_C$ (resp., $\mathtt{volume}_C$, $\mathtt{number}_C$, etc.) is equal to the sum of the $\mathtt{mass}_{D_i}$'s (resp., of the $\mathtt{volume}_{D_i}$'s, of the $\mathtt{number}_{D_i}$'s, etc.).

Finally, each variable v is assumed to satisfy the sign constraint $v \geqslant 0$.

The substitution (4) indicates that there should be neither egg nor armagnac in the adapted recipe. By contrast, there should be some bananas and kirsch but this piece of information can be entailed by the conservation equations. Therefore, the query is simply modeled by:

$$\mathtt{Q} = (\mathtt{mass}_{\mathtt{Egg}} = 0) \wedge (\mathtt{mass}_{\mathtt{Armagnac}} = 0) \tag{8}$$

The adaptation problem is now formalized: the source case is formalized by (5); the query is formalized by (8) and the domain knowledge is given by the conversion and conservation equations, and the sign constraints. Since the source case and the query are to be understood wrt the domain knowledge, the formulas for them are, respectively, $\mathtt{DK} \wedge \mathtt{Source}$ and $\mathtt{DK} \wedge \mathtt{Q}$. The result of the adaptation will be denoted by AdaptedCase.

Description of the adaptation process. Let $\{v_1, v_2, \ldots, v_n\}$ be the set of the variables used in `Source`, `Q` and `DK`. In the representation space based on the formalism used above, a particular recipe is represented by a tuple $x = (x_1, x_2, \ldots, x_n) \in \Omega$, where $\Omega = \Omega_1 \times \Omega_2 \times \ldots \times \Omega_n$ such that $\Omega_i = \mathbb{Z}$ if $\tau(v_i) =$ `integer` and $\Omega_i = \mathbb{R}$ otherwise ($\mathbb{R}$: set of real numbers, $\mathbb{Z}$: set of integers). Given φ, a conjunction of linear constraints, let $\mathcal{M}(\varphi)$ be the set of $x \in \Omega$ such that x verifies all the constraints of φ. The function $\varphi \mapsto \mathcal{M}(\varphi)$ provides a model-theoretical semantics to the logic of the conjunction of linear constraints: φ_1 entails φ_2 if $\mathcal{M}(\varphi_1) \subseteq \mathcal{M}(\varphi_2)$.

The principle of revision-based adaptation consists in a minimal modification of $\mathtt{DK} \wedge \mathtt{Source}$ so that it becomes consistent with $\mathtt{DK} \wedge \mathtt{Q}$. Such a minimal modification can be computed thanks to a belief revision operator based on a distance function d on Ω, meaning that the modification from an $x \in \Omega$ to an $y \in \Omega$ is measured by $d(x, y)$. Let $S = \mathcal{M}(\mathtt{DK} \wedge \mathtt{Source})$ and $Q = \mathcal{M}(\mathtt{DK} \wedge \mathtt{Q})$. The minimal modification from the source case to the query is therefore measured by $d^* = d(S, Q) = \inf_{x \in S, y \in Q} d(x, y)$. Thus, `AdaptedCase` is such that

$$\mathcal{M}(\mathtt{AdaptedCase}) = \{y \in Q \mid d(S, y) = d^*\}$$

where $d(S, y) = \inf_{x \in S} d(x, y)$.

Now, d is assumed to be a Manhattan distance function:

$$d(x, y) = \sum_{i=1}^{n} w_i |y_i - x_i|$$

where $w_i > 0$ is a weight associated to the variable v_i. Such a weight captures the effort of change for this variable. For example, if $v_i = \mathtt{volume}_{\mathtt{LemonJuice}}$ and $v_j = \mathtt{volume}_{\mathtt{Vodka}}$, then $w_i < w_j$ means that the adaptation process is less “reluctant” to change the volume of lemon juice than to change the volume of vodka.

Under this assumption, $\mathcal{M}(\mathtt{AdaptedCase})$ is the solution of the following optimization problem in y:

$$x \in \mathcal{M}(\mathtt{DK} \wedge \mathtt{Source}) \qquad y \in \mathcal{M}(\mathtt{DK} \wedge \mathtt{Q}) \tag{9}$$

$$\text{minimize } d(x, y) \tag{10}$$

The conjunctions of constraints (9) are linear but the objective function (10) is not. Now, it can be shown that the set of solutions to this problem coincides with the set of solutions to the following optimization problem in y:

$$\begin{array}{cc} x \in \mathcal{M}(\mathtt{DK} \wedge \mathtt{Source}) & y \in \mathcal{M}(\mathtt{DK} \wedge \mathtt{Q}) \\ \bigwedge_{i=1}^{n} z_i \geq y_i - x_i & \bigwedge_{i=1}^{n} z_i \geq x_i - y_i \end{array}$$

$$\text{minimize } \sum_{i=1}^{n} w_i z_i$$

which is linear, and thus can be solved with classical operational research techniques. It is noteworthy that if every variable is continuous, then this optimization problem is polynomial, otherwise, it is a mixed integer linear optimization, known to be an NP-hard problem. In practice, the more variables are integers, the more it will require computing time; thus, if a variable range is big enough, it may be more appropriate to consider it as real. The heuristic we have chosen is as follows. If, for a type of food F, it appears in all the recipes of the case base as units, then $\tau(\texttt{number}_F) = \texttt{integer}$.

When this linear problem is solved, this gives a solution to the query, expressed with all the n variables. From a human-interface viewpoint, some of these variables should not be displayed. For example, if an ingredient is given by its volume in the source recipe, then it should not be given as a mass in the adapted case. Since DK relates masses to volumes, there is no loss of information.

With the example presented above, the result is as follows:

$$\begin{aligned}\texttt{AdaptedCase} \equiv \texttt{DK} \\ &\wedge (\texttt{volume}_{\texttt{Kirsch}} = 9) \wedge (\texttt{volume}_{\texttt{Rum}} = 25) \wedge (\texttt{volume}_{\texttt{Milk}} = 50) \\ &\wedge (\texttt{number}_{\texttt{Banana}} = 2) \wedge (\texttt{mass}_{\texttt{GranulatedSugar}} = 96) \\ &\wedge (\texttt{volume}_{\texttt{FreshCream}} = 290)\end{aligned}$$

It can be noticed that AdaptedCase entails $\texttt{DK} \wedge \texttt{Q}$, which was expected. For this example, the following weights have been chosen assuming that more a variable correponds to a general concept more its associated weight has to be large:

$$\begin{array}{lll} w_{\texttt{volume}_{\texttt{Food}}} = 100 & w_{\texttt{sugar}_{\texttt{Food}}} = 50 & w_{\texttt{alcohol}_{\texttt{Food}}} = 50 \\ w_{\texttt{volume}_{\texttt{Brandy}}} = 5 & w_{\texttt{mass}_{\texttt{EggOrEquivalent}}} = 10 & \end{array}$$
$$\text{and } w_v = 1 \text{ for any other variable } v$$

Translated back in an informal way, this gives:

AdaptedCase =

Recipe “Eggnog” (10 glasses) after adaptation
9 cℓ of kirsch, 25 cℓ of rum, half a liter of milk, 2 bananas, 96 g of sugar, 290 cℓ of fresh cream

This result illustrates the quantity compensations done by the adaptation: the quantity of sugar has been lowered because bananas are sweeter than eggs and the volume of kirsch is higher than the volume of armagnac in the source recipe, because the degree of alcohol is lower for armagnac than for kirsch.

4 Sandwich challenge

The *sandwich challenge* is addressed with the 2014 Taaable system [9], which is efficient for the ingredient susbtitution step. The preparation procedure of the adapted recipe uses, in the same order, the steps used in the source recipe, because the ontology-based substitution procedure of Taaable favors the substitution of ingredients of the same type (e.g., a sauce by a sauce). So, the order of the ingredients in the adapted recipe will be the same as in the source recipe.

To adapt the textual preparation of the recipe, the text occurrences of the replaced ingredients are substituted with the replacing ingredients. A set of rules allows to identify plurals of the removed ingredient in the text, and replace them with the plural form of the replacing ingredients. For example, when replacing *mayo* with *mustard*, “Apply *mayo* on one slice, tomato sauce on the other.” is adapted to “Apply *mustard* on one slice, tomato sauce on the other.”

5 Conclusion

This paper has presented the two systems developed by the TAAABLE team for its participation to the 2015 CCC. The two systems are based on the previous version of TAAABLE, extended with two new approaches: a FCA approach to guide ingredient substitution, and an adaptation of the ingredient quantities based on a mixed linear optimization. The work presented here still needs a thorough evaluation: ongoing work addresses this issue, following the methodology introduced in [2].

References

1. A. Cordier, V. Dufour-Lussier, J. Lieber, E. Nauer, F. Badra, J. Cojan, E. Gaillard, L. Infante-Blanco, P. Molli, A. Napoli, and H. Skaf-Molli. Taaable: a Case-Based System for personalized Cooking. In S. Montani and L. C. Jain, editors, *Successful Case-based Reasoning Applications-2*, volume 494 of *Studies in Computational Intelligence*, pages 121–162. Springer, 2014.
2. E. Gaillard, J. Lieber, E. Nauer, and A. Cordier. How Case-Based Reasoning on e-Community Knowledge Can Be Improved Thanks to Knowledge Reliability. In *Case-Based Reasoning Research and Development*, volume 8765, pages 155 – 169, Cork, Ireland, Ireland, September 2014. L. Lamontagne and E. Plaza.
3. E. Gaillard, L. Infante-Blanco, J. Lieber, and E. Nauer. Tuuurbine: A Generic CBR Engine over RDFS. In *Case-Based Reasoning Research and Development*, volume 8765, pages 140 – 154, Cork, Ireland, September 2014.
4. E. Gaillard, J. Lieber, and E. 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, 2011.
5. B. Ganter and R. Wille. *Formal Concept Analysis: Mathematical Foundations.* Springer, Berlin, 1999.
6. C. Carpineto and G. Romano. Effective Reformulation of Boolean Queries with Concept Lattices. In T. Andreasen, H. Christiansen, and H. Legind Larsen, editors, *Flexible Query Answering Systems, Third International Conference (FQAS'98)*, volume 1495 of *LNCS*, pages 83–94. Springer, 1998.
7. C. Carpineto and G. Romano. Order-Theoretical Ranking. *Journal of the American Society for Information Science*, 51(7):587–601, 2000.
8. J. Cojan and J. Lieber. Applying Belief Revision to Case-Based Reasoning. In H. Prade and G. Richard, editors, *Computational Approaches to Analogical Reasoning: Current Trends*, volume 548 of *Studies in Computational Intelligence*, pages 133 – 161. Springer, 2014.
9. E. Gaillard, J. Lieber, and E. Nauer. Case-Based Cooking with Generic Computer Utensils: Taaable Next Generation. In *Proceedings of the ICCBR 2014 Workshops*, number pp 89-100, page 254, Cork, Ireland, 2014. D. B. Leake and J. Lieber.